UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

PROYECTO FIN DE CARRERA

DISECTOR DE PROTOCOLOS LDAP

GUILLERMO MUÑOZ MOZOS

ABRIL 2014

DISECTOR DE PROTOCOLOS LDAP

AUTOR: Guillermo Muñoz Mozos
TUTOR: Javier Aracil Rico

HPCN GROUP
Dpto. Tecnología Electrónica y de Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Abril de 2014

Resumen

El protocolo LDAP (Lightweight Directory Access Protocol) es un protocolo a nivel de aplicación que permite el acceso a un servicio de directorio ordenado y distribuido para buscar diversa información en un entorno de red. LDAP también se considera una base de datos a la que pueden realizarse consultas.

LDAP es ampliamente utilizado por cualquier compañía u organización que requiera acceso a una red utilizando información de autenticación. Además de información de contactos o personas, es capaz de acceder a certificados encriptados, punteros a impresoras y otros servicios de una red, y proporcionar un acceso único cuando una contraseña para un usuario está compartida entre determinados servicios.

LDAP es apropiado para cualquier información de tipo directorio, donde la norma es acceder de forma rápida a dicha información y actualizarla de manera poco frecuente.

En el presente proyecto fin de carrera se diseña un disector de las versiones 2 y 3 de este protocolo, utilizando librerías de código abierto desarrolladas en lenguaje de programación C por el grupo de investigación HPCN de la Universidad Autónoma de Madrid.

En primer lugar, se realiza un estudio del estado del arte de dicho código aplicando las funciones básicas de un disector.

Posteriormente, se analiza el comportamiento del protocolo con el fin de desarrollar un disector que analice todos sus campos para detectar posibles fallos en la comunicación, así como para encontrar las causas de dichos fallos y estudiar las posibles soluciones.

Finalmente, se realizan pruebas de velocidad de procesado, consumo de memoria y efectividad de detección.

Esto permite desarrollar un disector altamente eficiente que permite manejar tramas de gran tamaño en poco tiempo y asegurar que se detecta adecuadamente el fallo en la transmisión del protocolo a través de Internet.

Palabras clave

LDAP, Protocolo, Nivel de Aplicación, Directorio, Base de Datos, Cliente – Servidor, Código Abierto, C, Eficiente.

Abstract

The Lightweight Directory Access Protocol (LDAP) is an application protocol for accessing to an organized and distributed directory service to find network information. Also its considered a database which queries can be performed.

LDAP is widely used for any company or organization that requires network access using authentication information. It is possible to access to encrypted certifies, pointers to printers and other network services in addition of contact or people information. Also it provides a single access when an user password it is shared between services.

LDAP is appropriated for any directory kind information for accessing and not refreshing frequently.

In the present project, it is developed a dissector for version 2 and 3 of this protocol, using open source code based on C programming language designed by the HPCN Researching Group of the University Autonoma of Madrid.

First, it is performed a study of the state of the art of this code applying the main functions of a basic dissector.

Then it is analyzed the behavior of the protocol with the purpose of developing a dissector that analyzes all the fields for detecting possible communication fails, as well as finding the causes and finding the possible solutions.

Finally, it is performed speed execution, memory intake and detection reliability. This allows to develop a high quality accurate dissector to handle big size frames in few time and to ensure that it is detected appropriately the fail of the protocol communication through Internet.

Keywords

LDAP, Protocol, Application Protocol, Directory, Database, Client-Server Model, Open Source Code, C, High Quality, Accurate.

Agradecimientos

En primer lugar me gustaría agradecer la oportunidad de entrar en el grupo de investigación HPCN brindada por mi tutor Javier Aracil Rico, así como su interés por el aprendizaje de mis conocimientos y su gran ayuda.

Doy gracias también al resto de profesorado que, desde el interés por la enseñanza, han conseguido crear en mí un orgulloso Ingeniero de Telecomunicación.

Puesto que es el final de una carrera, la referencia a todas las personas que me han acompañado a lo largo de estos años es obligatoria. Todos los amigos que han compartido sus días conmigo son dignos de mención ya que sólo me han servido para mejorar como persona y estudiante, pero si tengo que destacar a los más importantes, esos son Luis Rubio, Jesús Vázquez, Pedro Lorenzo, Javier Del Sol y Sergio Carrero.

Por último y, lo más importante de todo, agradecer a mis padres la educación y valores que me han impartido ya que sin ellos no hubiera conseguido llegar hasta aquí.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1. Introducción

1.1 Motivación

La motivación de este proyecto consiste en desarrollar un disector de las versiones 2 y 3 del protocolo LDAP utilizando un lenguaje de programación basado en código abierto desarrollado por el departamento HPCN de la Universidad Autónoma de Madrid.

Este disector permite analizar trazas de varios TBytes con una alta velocidad de procesado, permitiendo realizar un estudio del comportamiento del protocolo a lo largo del tiempo. De esta manera, se podrán detectar problemas, encontrar las causas y plantear soluciones que mejoren la operativa del servicio.

Un problema en la comunicación del protocolo puede suponer grandes pérdidas para entidades como bancos u organizaciones que necesiten la transmisión del protocolo LDAP al utilizar aplicaciones que funcionen en Internet basadas en TCP/IP.

En este proyecto se consigue un programa optimizado para detectar dichos problemas en corto espacio de tiempo y sin consumir demasiados recursos en el sistema que lo utilice.

No sólo puede detectar problemas en la comunicación sino que obtiene los campos relevantes del protocolo aportando un nivel de detalle que permite al usuario conocer las causas de dichos problemas y encontrar las soluciones al respecto.

1.2 Objetivos

El objetivo del proyecto es diseñar un programa desarrollado en lenguaje de programación en C que sea capaz de analizar trazas reales de enorme tamaño de manera altamente optimizada.

Este disector es capaz de analizar miles de mensajes transmitidos permitiendo el estudio del protocolo desde que se establece conexión hasta que se termina la sesión o la comunicación. Se detectaría un posible fallo en la comunicación ya que se extraen los campos relativos al problema y, se podrían analizar las causas observando la salida de los campos con el fin de aportar soluciones de forma rápida y sencilla.

Todo esto se estudia a lo largo del proyecto optimizando el programa gracias al resultado de las pruebas y su contraste con lo esperado.

Para ello, se analizan múltiples tramas reales procedentes de entidades conocidas, así como se prueba el disector en servidores alojados en Méjico con el fin de mejorar la efectividad del disector y obtener un programa de altas prestaciones.

A lo largo del proyecto, se realizan pruebas para comprobar el rendimiento del programa y mejorar sus prestaciones. Desde pruebas de consumo de memoria, tiempo de ejecución de dicho programa hasta pruebas en discos de lectura o discos de lectura y escritura.

Tras cumplir el objetivo de estas pruebas se consigue finalmente desarrollar un disector capaz de detectar errores en la comunicación del protocolo, en mucho menor tiempo que otras herramientas de disección como Tshark y, utilizando librerías propias de la Universidad.

Esto abre una vía de estudio del protocolo LDAP a fondo ya que se obtienen todos los campos relevantes del protocolo de forma rápida y sencilla y, se pueden obtener los detalles del problema de la comunicación con el fin de solucionar el motivo que lo ha causado.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- Capítulo 1: Introducción y motivación del proyecto.
- Capítulo 2: Estado del arte del protocolo LDAP.
- Capítulo 3: Diseño del disector.
- Capítulo 4: Desarrollo del programa.
- Capítulo 5: Integración, pruebas y resultados.
- Capítulo 6: Conclusiones y trabajo futuro.

2. Estado del arte

El estudio del estado del arte en este proyecto se centra única y exclusivamente en el protocolo LDAP. Es lógico, por tanto, que se muestre el funcionamiento del protocolo explicando al máximo detalle posible su comportamiento de cara a entender las bases que rigen la necesidad de realizar un disector adecuado.

2.1 LDAP (Lightweight Directory Access Protocol)

LDAP, es un protocolo de Internet que utilizan programas y emails para buscar información en un servidor. Utiliza directorios basados en el estándar de servicios de directorios X.500.

LDAP fue diseñado en la Universidad de Michigan para adaptar el complejo sistema de directorios X.500 al Internet actual. X.500 era muy complejo para utilizarse en Internet, así que se creó LDAP para dar este tipo de servicio al cliente.

X.500 es un conjunto de estándares de redes de ordenadores de la ITU-T sobre servicios de directorio, entendidos estos como bases de datos de direcciones electrónicas. El estándar se desarrolló conjuntamente con la ISO como parte del Modelo de interconexión de sistemas abiertos, para usarlo como soporte del correo electrónico X.400.

Los protocolos definidos por X.500 son el protocolo de acceso al directorio (DAP), el protocolo de sistema de directorio, el protocolo de ocultación de información de directorio, y el protocolo de gestión de enlaces operativos al directorio.

LDAP surgió por la necesidad de utilizar el protocolo de acceso a directorio (DAP) sobre aplicaciones que se basaran en el modelo TCP/IP como sistema de gestión de equipos para comunicarse en red.

Una vez entendido el origen del protocolo, se explica a continuación su funcionamiento.

Cada programa de email tiene un libro de direcciones personal, pero ¿cómo buscar una dirección para alguien que nunca te ha mandado un email?. ¿Cómo una organización puede tener una libreta telefónica actualizada y

centralizada a la que todo el mundo puede acceder de manera rápida y sencilla?.

Todo esto se consigue estructurando la información en entradas a modo de árbol. Una entrada es una colección de atributos que tienen un único campo global y distinguido. cada atributo de las entradas tiene un tipo y uno o más valores. Los valores tipo suelen ser cadena de caracteres aunque su valor depende del tipo de atributo al que se refiera. Por ejemplo, un atributo tipo "cn" puede contener el valor "Babs Jensen" pero, un atributo tipo "mail" puede contener el valor "babs@example.com". Otro valor de un tipo "messageID" sin embargo puede ser "0x00d5" en hexadecimal que en decimal sería "213".

A continuación se ilustra el sistema de organización de la información en entradas.

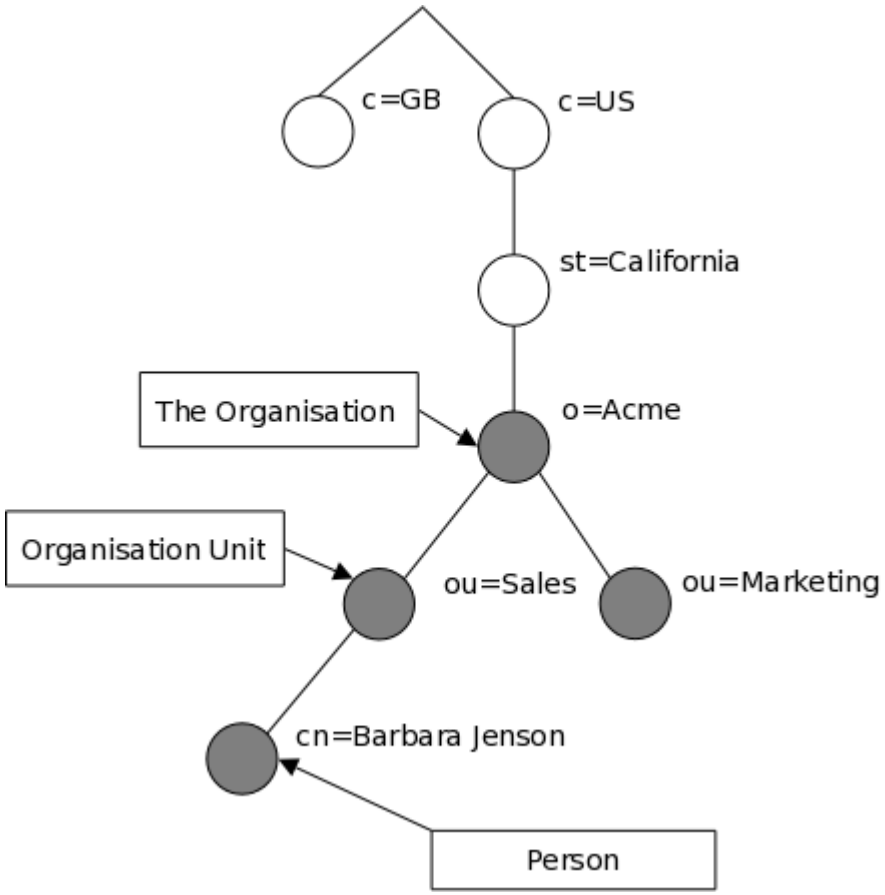

Figura 2.1-1: Organización de sistema por entradas

La manera más popular de organizar los árboles de entradas a los directorios es utilizar nombres de dominio asociados a Internet (DNS). Aquí un ejemplo de una organización que utilizara dominio ".com".

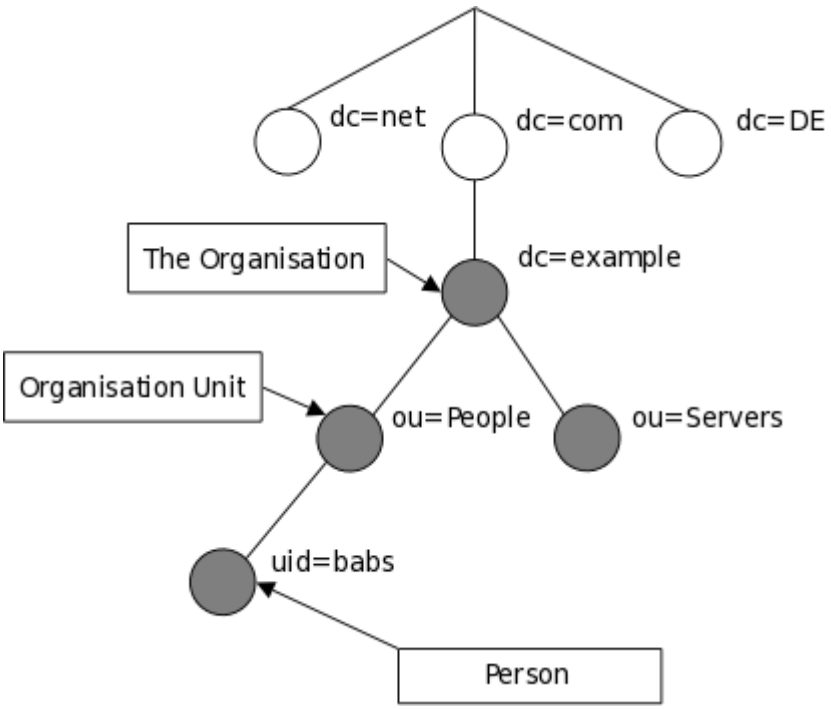

Figura 2.1-2: Sistema de organización de entradas con DNS

Además, LDAP permite controlar los atributos que son requeridos y permitidos en una entrada mediante el atributo "objectClass". El valor de este atributo determina las reglas que la entrada deberá seguir.

Una entrada es referenciada mediante su nombre distinguido (DN), que se construye con el nombre de la propia entrada (denominado Nombre Distinguido Relativo o RDN) y, el nombre de las entradas que lo preceden concatenado. Por ejemplo, la entrada para Barbara Jenson del ejemplo anterior tiene un RDN "uid=babs" y un DN "uid=babs,ou=People,dc=example,dc=com".

Los programas cliente LDAP pueden preguntar a los servidores LDAP para buscar entradas de una manera muy amplia. Los servidores LDAP guardan todos los datos de sus entradas, y los filtros que podrían ser utilizados para seleccionar sólo la persona o el grupo que se considere necesario. Ejemplo de una búsqueda LDAP:

"Buscar el árbol de directorio completo que contenga "dc=example,dc=com" para aquellas personas que cuyo nombre sea "Barbara Jensen" y, devolver la dirección de correo de cada entrada encontrada."

Otro ejemplo de búsquedas ilustrado en imágenes.

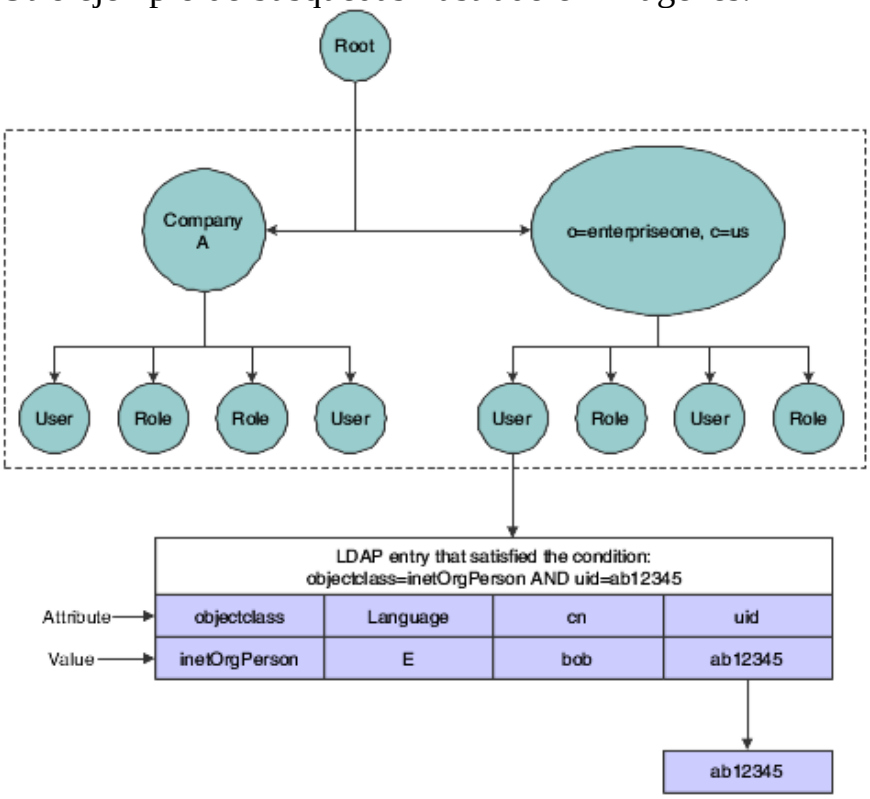

Figura 2.1-3: Búsqueda de entradas LDAP

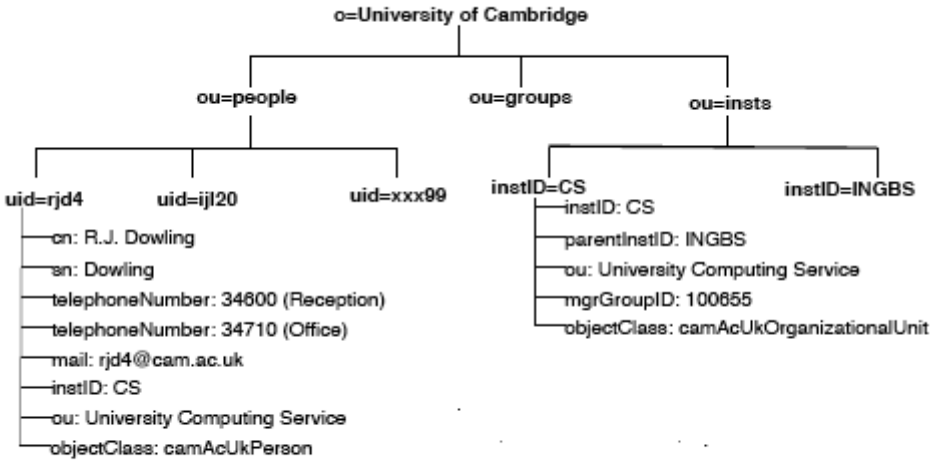

Figura 2.1-4: Jerarquía de entradas Universidad de Cambridge

El protocolo proporciona un mecanismo para que el cliente se autentifique o pruebe su identidad en servidor de directorios, protegiendo la información que contiene el servidor. LDAP también soporta servicios de seguridad de datos como TLS.

LDAP, además de información de contactos o personas, es capaz de acceder a certificados encriptados, punteros a impresoras y otros servicios de una red, y proporcionar un acceso único cuando una contraseña para un usuario está compartida entre determinados servicios. Aquí un ejemplo de un directorio relativo a una Universidad cuya estructura es más compleja.

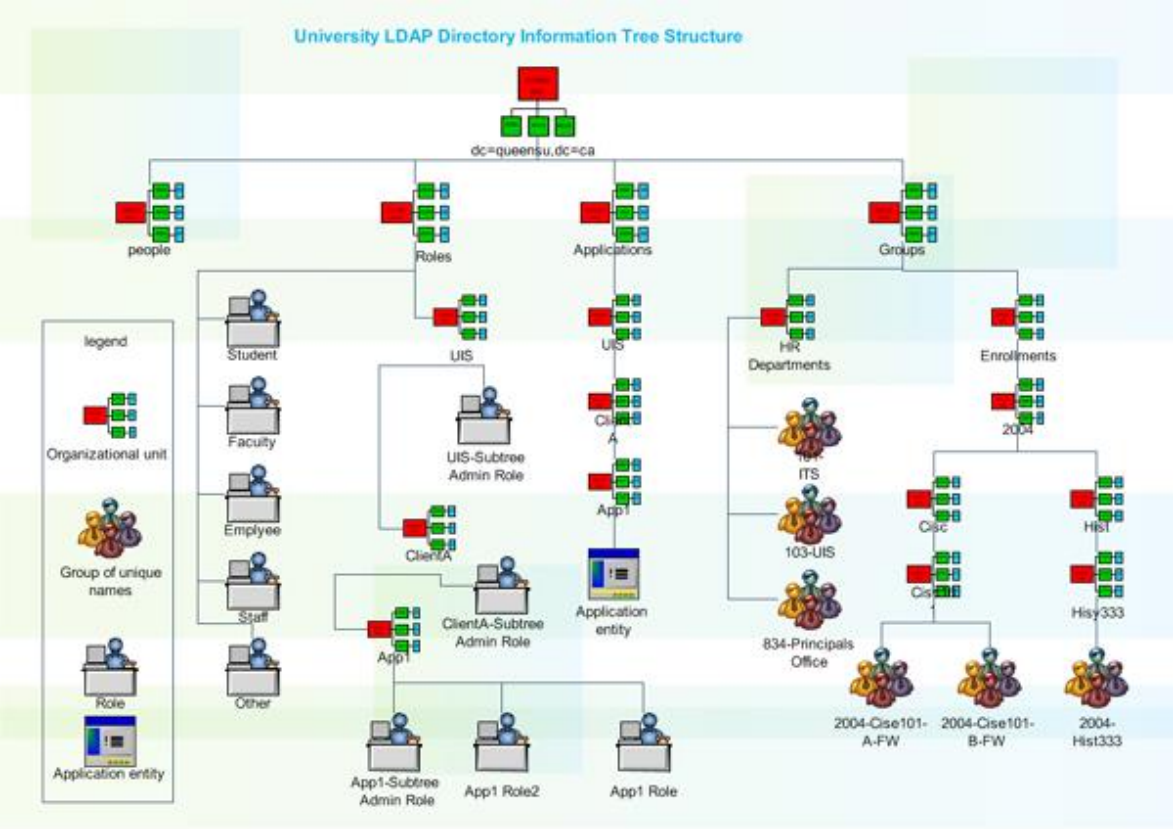

Figura 2.1-5: Árbol de entradas con estructura compleja

LDAP es apropiado para cualquier información de tipo directorio, donde la norma es acceder de forma rápida a dicha información y actualizarla de manera poco frecuente.

Como protocolo que es, LDAP no define cómo los programas trabajan desde el lado del cliente o el servidor. Define el "lenguaje" utilizado de los programas clientes cuando interactúan con el servidor (de servidor a servidor también).

En el lado del cliente, un cliente podría ser un programa tipo email, un navegador de impresora, o un libro de direcciones. El servidor debería de hablar "lenguaje" LDAP, o tener otros métodos para enviar y recibir datos donde LDAP sería un método añadido.

A continuación, un ejemplo de la evolución de la comunicación cliente servidor desde X.500 hasta la actualidad con LDAP.

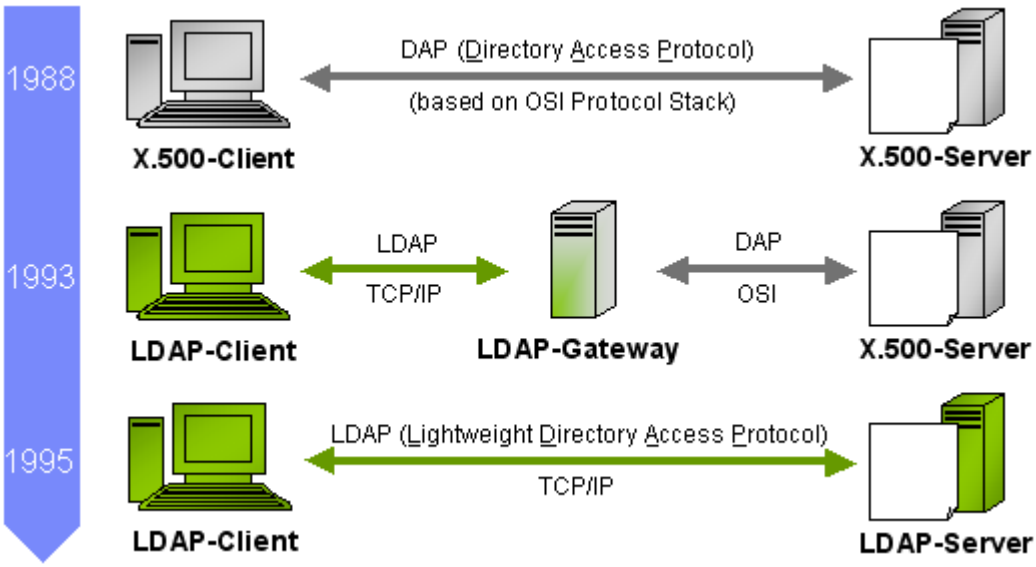

Figura 2.1-6: Evolución X.500 - LDAP

LDAP por sí sólo no incluye seguridad o encriptación, utiliza mecanismos de encriptación conectados al servidor LDAP como "encrypted SSL".

LDAP también define:

- Permisos: otorgados por el administrador que permite sólo a ciertas personas acceder a la base de datos LDAP, y opcionalmente guardar cierto dato privado.

- Esquemas: es una manera de describir el formato y los atributos de los datos del servidor. Los atributos normales como el nombre la dirección email tienen que ser heredados de uno de los esquemas estándar, que parten del protocolo de directorios X.500.

LDAP existe a tres tipos de niveles: Servidores públicos de gran tamaño, servidores organizativos en Universidades y corporaciones o, servidores pequeños para grupos de trabajo.

A continuación, se explica en detalle tanto la mecánica del protocolo como los campos que lo constituyen.

2.1.1 EL PROTOCOLO

El modelo general adoptado por este protocolo son operaciones cliente-servidor. En este modelo, un cliente transmite una petición describiendo la operación para que el servidor la procese. El servidor es el responsable de procesar la operación e implementar lo necesario para ejecutar dicha operación en el directorio.

Típicamente, el servidor devuelve una respuesta conteniendo el dato correspondiente a la petición realizada previamente por el cliente.

Las operaciones del protocolo LDAP son generalmente independientes una de otra. Cada operación se procesa como una acción atómica, dejando el directorio en un estado consistente.

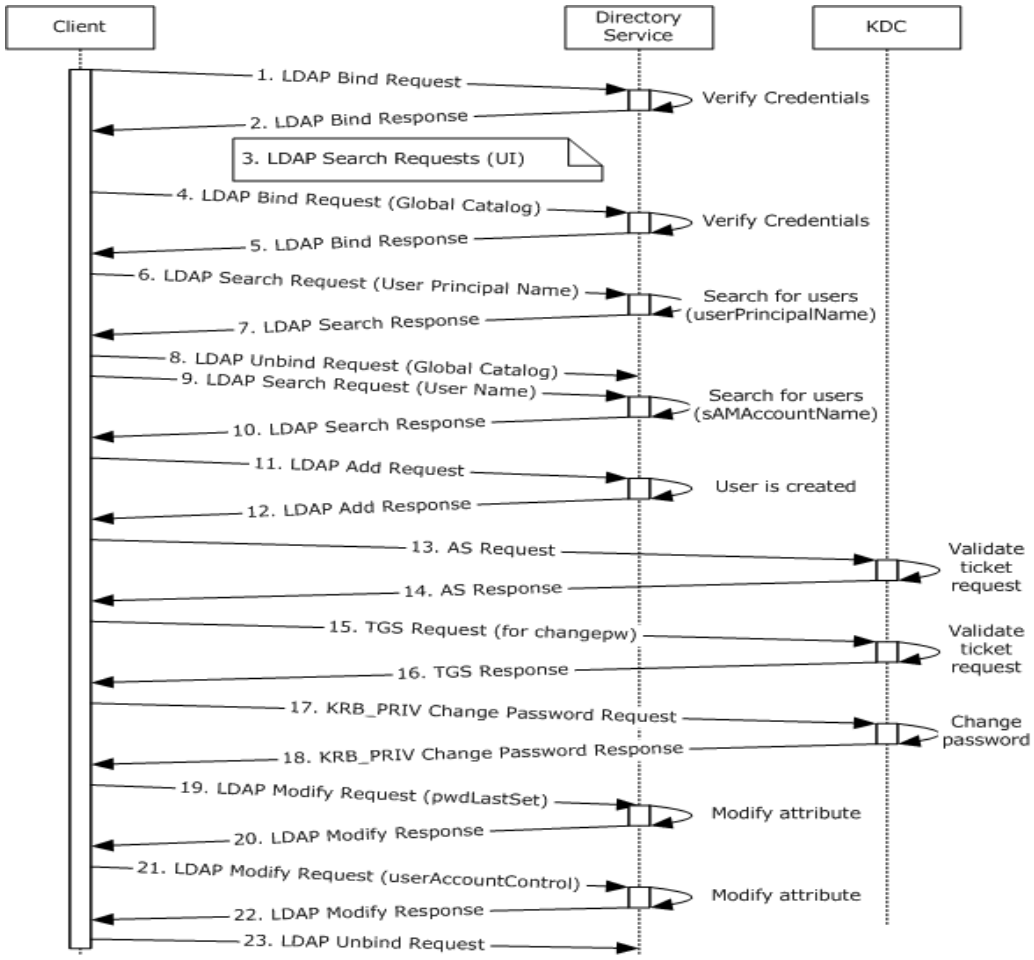

Figura 2.1-8: Comunicación LDAP

Como los servidores deben devolver respuestas de peticiones realizadas por el cliente definidas en el protocolo, no se requiere de un comportamiento síncrono por ninguna de las dos partes. Tanto las peticiones como las respuestas de múltiples operaciones, generalmente no deben seguir orden alguno. Si se requiriera, el comportamiento síncrono debería de ser controlado por las aplicaciones cliente.

El núcleo de las operaciones del protocolo puede ser mapeado como un subset del protocolo de directorios X.500 (Directory Abstract Service X.511). Sin embargo, no hay un mapeo uno-a-uno entre las operaciones LDAP y las operaciones del protocolo X.500. Es en el servidor donde se tiene que implementar una vía para realizar múltiples peticiones del protocolo X.500 de un servicio en una única petición LDAP.

2.1.2 OPERACIONES Y LA CAPA LDAP MESSAGE

Las operaciones del protocolo LDAP forman parte de la capa del mensaje LDAP. Cuando la conexión se cierra, cualquiera operación incompleta en cualquiera capa del mensaje LDAP es abandonada o es completada sin transmitir la respuesta. En este caso, si de una operación de actualización incompleta se tratara, no se debería asumir que la operación ha sido correcta o fallida.

La función del mensaje LDAP (LDAPMessage) es proporcionar una estructura con los mismos campos requeridos para todas las peticiones del protocolo. Los campos en común son la identificación de mensaje (messageID) y el campo tipo control (controls).

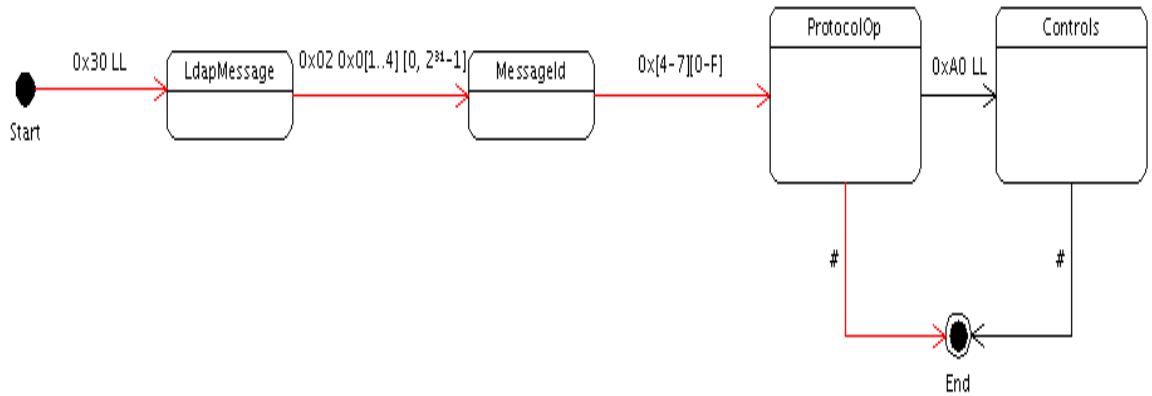

Figura 2.1-7: LDAPMessage

Si el servidor recibe un mensaje LDAP (LDAPMessage) del cliente en el que la secuencia de dicho mensaje (LDAPMessage SEQUENCE) no se ha podido reconocer, el campo de identificación de mensaje (messageID) no se ha podido procesar, la etiqueta del tipo de operación de protocolo (protocolOp) no se ha reconocido como petición o, las estructuras o longitudes de los campos relativos a la información del mensaje, se han procesado de forma incorrecta, entonces el servidor debería devolver un estado de desconexión.

En este estado el código de operación (resultCode) debería ser establecido a error de protocolo (protocolError) y, se debería terminar inmediatamente la sesión LDAP.

2.1.3 MESSAGEID

Todos los mensajes LDAP (LDAPMessage) contienen el valor del número de identificación de mensaje (messageID) de la petición correspondiente al mensaje LDAP (LDAPMessage).

Deberá tener un valor distinto de cero y diferente de cualquier otro número de identificación de mensaje (messageID) de cualquier petición en progreso de la misma sesión LDAP. El valor de cero está reservado para la notificación de mensaje sin solicitud.

El cliente no deberá enviar una petición con el mismo número de identificación de mensaje (messageID) que una petición realizada anteriormente en la misma sesión LDAP salvo que se pueda determinar con seguridad que el servidor ya no está dando servicio a dicha petición. De otra manera, el comportamiento es impredecible.

2.1.4 TIPOS DE CADENAS STRING LDAP

La cadena LDAP (LDAPString) es la notación apropiada que indica que, aunque las cadenas de caracteres de tipo octeto (OCTET STRING) estén codificadas como ASN.1 , el set de caracteres es utilizado y codificado siguiendo el algoritmo UTF-8.

El OID de tipo LDAP (LDAPOID) es la notación correspondiente a los valores permitidos de una cadena de caracteres codificada en UTF-8 de la representación del identificador de un objeto.

Se adjunta un esquema para una organización con OID tipo LDAP cuya base es “1.1”:

OID	Assignment
1.1	Organization's OID
1.1.1	SNMP Elements
1.1.2	LDAP Elements
1.1.2.1	AttributeTypes
1.1.2.1.1	x-my-Attribute
1.1.2.2	ObjectClasses
1.1.2.2.1	x-my-ObjectClass

Tabla 2.1-1: Tabla OIDs de LDAP

2.1.5 NOMBRE DISTINGUIDO Y RELATIVO

Se muestran los campos de tipo cadena de caracteres que representan el nombre distinguido y el nombre relativo distinguido. Se han mostrado anteriormente ejemplos de su uso y aplicación.

El LDAPDN es el nombre que define la representación de un nombre distinguido (DN).

El RelativeLDAPDN es la representación de un nombre distinguido relativo (RDN).

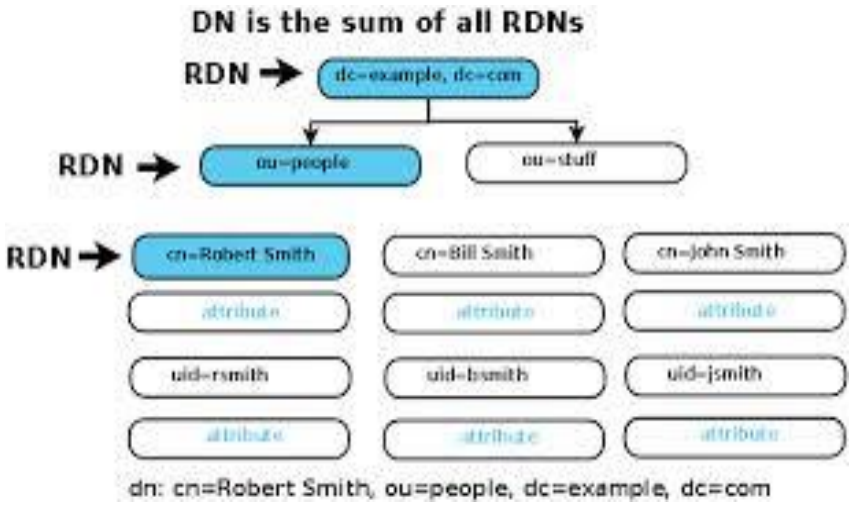

Figura 2.1-9: RDN y DN nombres de LDAP

2.1.6 ATRIBUTOS

El campo que contiene la descripción de los atributos referenciados en cada mensaje LDAP se denomina "AttributeDescription" y rige la siguiente sintaxis:

Un campo del tipo valor de atributo (AttributeValue) es una cadena de tipo octeto (OCTET STRING) que contiene el valor de atributo codificado. Dicho valor está codificado de acorde a la especificación LDAP de su sintaxis correspondiente.

Ya que el valor del campo tipo atributo puede tener una sintaxis arbitraria y es posible que no se pueda traducir su valor, las implementaciones no deberán mostrar o intentar decodificar ningún valor del atributo si la sintaxis no es conocida. Dicha implementación debería intentar acceder a descubrir el subschema del código de la entrada y recuperar la descripción del campo de tipo atributo (attributeTypes).

Los clientes sólo deberán enviar los valores del atributo en una petición que sean válidos acordes a la sintaxis definidas de los atributos.

2.1.7 RESULTADO LDAP

El resultado LDAP (LDAPResult) es el constructor utilizado en el protocolo para retornar éxito o fallo desde servidores a clientes. Realizando varias peticiones, los servidores retornarán respuestas conteniendo los elementos encontrados en el resultado LDAP (LDAPResult) para indicar el estado final de la petición de operación del protocolo.

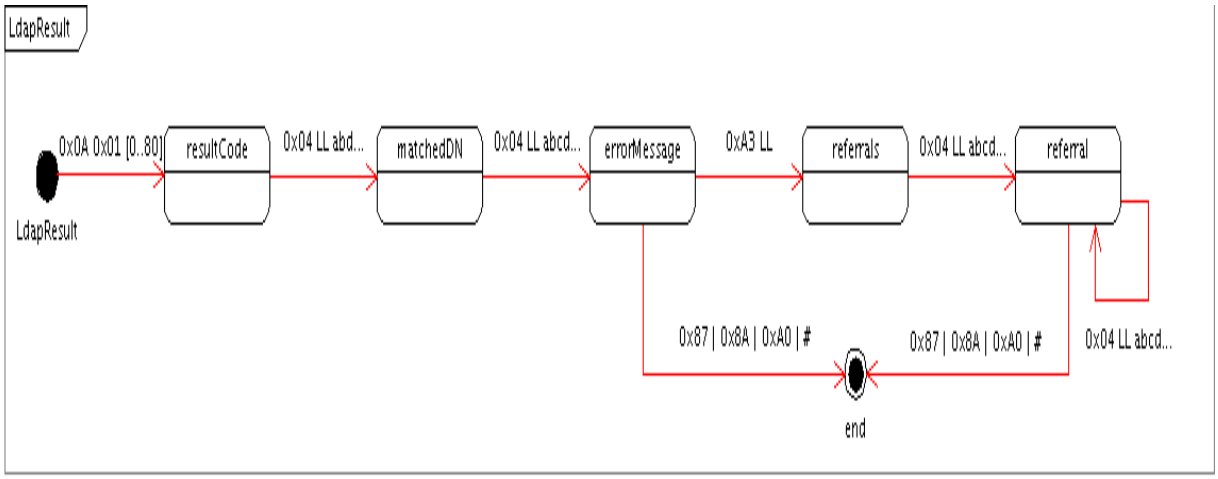

Figura 2.1-10: LDAPResult

Figura 2.1-11: Trama BindResponse

Si el servidor detecta múltiples errores en una operación, sólo un código de resultado es devuelto. El servidor deberá por tanto devolver el código que mejor indique la naturaleza del error. El servidor deberá devolver códigos de operación (resultCode) substitutivos para prevenir revelaciones sin autorización.

2.1.8 CONTROLES

Los controles (controls) otorgan un mecanismo donde la semántica y los argumentos de una operación LDAP podría ser extendida. Un único mensaje LDAP puede tener uno o más controles. El control sólo afecta a la semántica del mensaje al que va relacionado.

Los controles enviados por los clientes son denominados 'controles de petición' (request controls) y, aquellos enviados por los servidores son denominados 'controles de respuesta' (response controls).

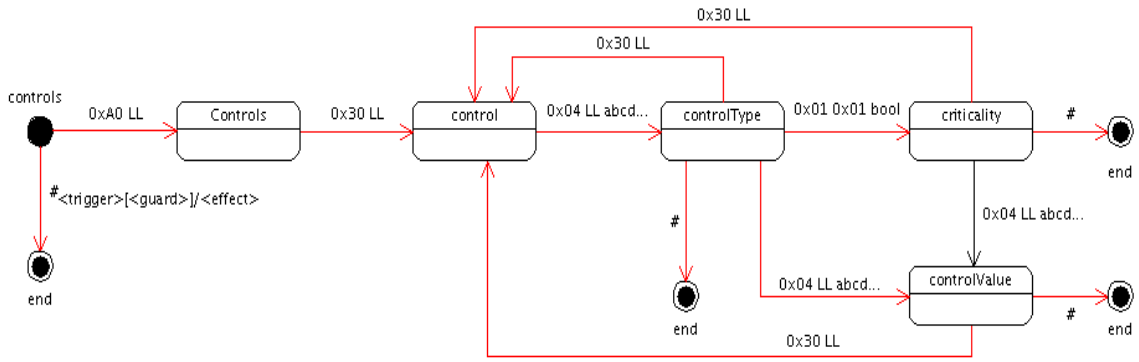

Figura 2.1-12: Controls

El tipo de control (controlType) es una representación decimal por puntos de un identificador de objeto que únicamente identifica al control. Algunas veces, el control de respuesta solicitado por el control de petición comparten el valor de tipo de control (controlType).

El campo de criticidad (criticality) sólo tiene significado en mensajes con control derivado a mensajes de petición (excepto UnbindRequest). Para controles relacionados con mensajes de respuesta y mensajes tipo UnbindRequest, el campo de criticidad (criticality) debe ser falso (FALSE) y, deberá ser ignorado por su par de protocolo de recepción. Un valor de verdadero (TRUE) indica que es inaceptable procesar la operación sin aplicar la semántica del control.

El valor de control (controlValue) debería contener información asociada al tipo de control (controlType). Su formato está definido por la especificación del control. La implementación deberá estar preparada para manejar contenidos de valor arbitrario del valor de control (controlValue).

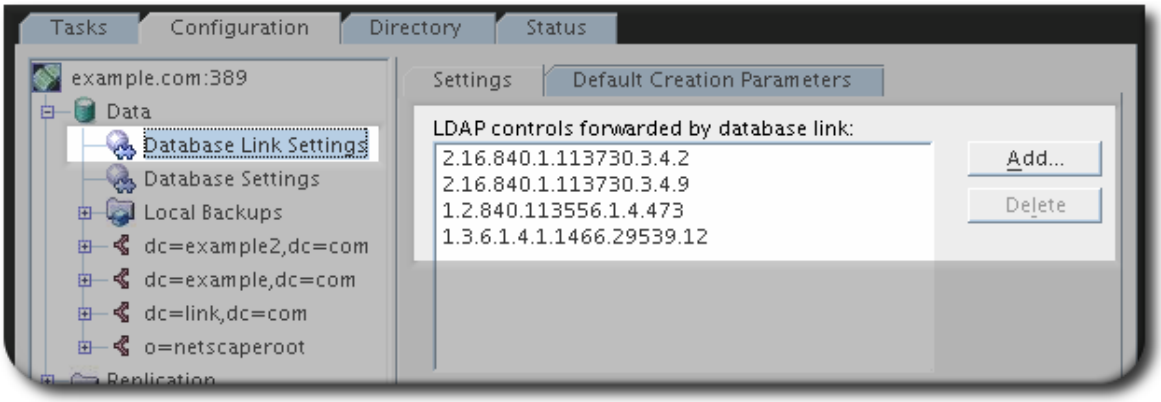

Figura 2.1-13: Controles en LDAP como base de datos

2.1.9.1 OPERACIONES LDAP: BINDREQUEST

La función de una operación de enlace es permitir información de autenticidad para ser compartida entre el cliente y el servidor. En resumen, es la operación de autentificación.

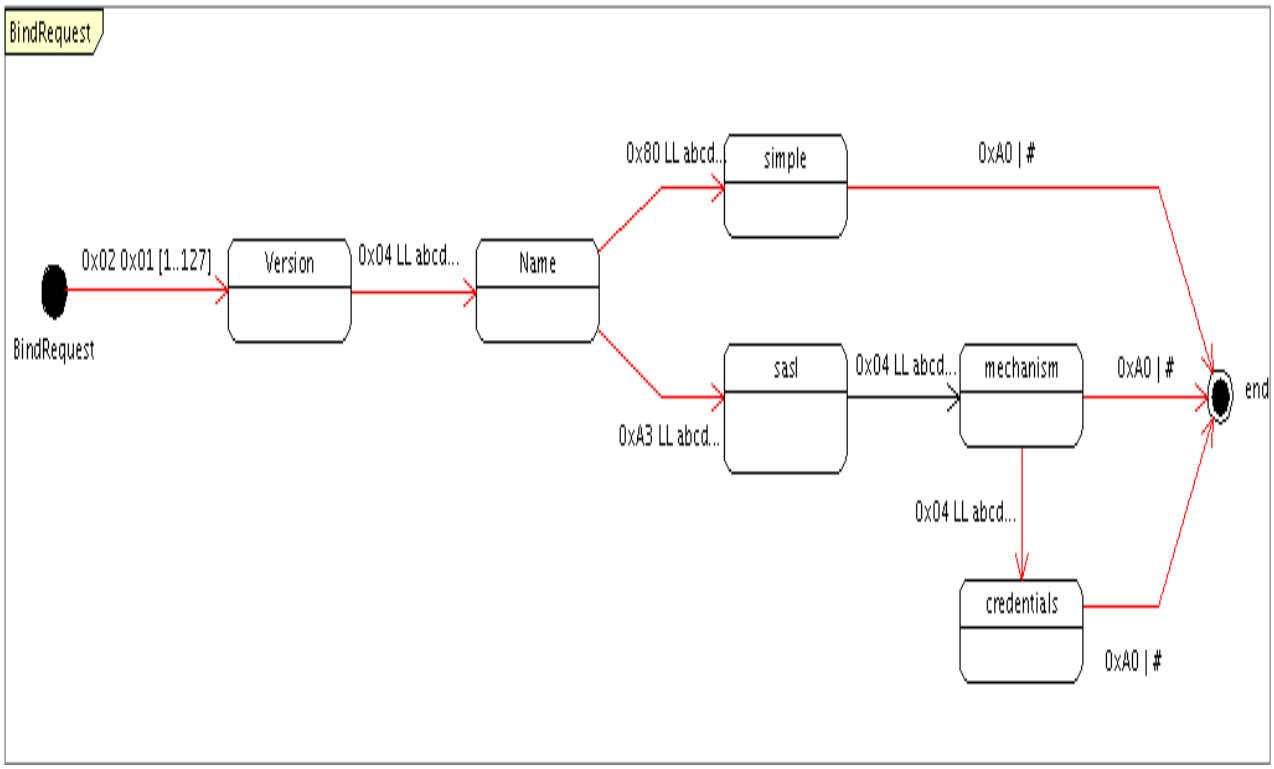

Figura 2.1-14: BindRequest

Antes de procesar una operación de enlace (BindRequest), todas las operaciones incompletas deberán completarse o ser abandonadas. Después, el servidor se dispone a autentificar al cliente en uno o varios pasos de enlace. Para ello, se debe devolver un valor de respuesta de enlace (BindResponse). De la misma manera, los servidores no deberían procesar o responder otras peticiones mientras procesan la actual petición.

Si el cliente no se consigue enlazar antes de enviar una petición y recibe un código de operación de error (operationsError) a esa petición, debería volver a mandar otra petición de enlace (bindRequest). Si esto falla también o el cliente elige no enlazarse en la sesión actual de LDAP, deberá restablecer la conexión y, volver al principio de la petición de enlace (bindRequest).

Los clientes podrían mandar múltiples peticiones de enlace para cambiar la autentificación y/o los credenciales de seguridad asociados o para completar un proceso de enlace de manera múltiple. La autentificación de los enlaces anteriores tiene que ser ignorada.

Para los casos de autentificación de mecanismo de SASL, será necesario para el cliente invocar múltiples veces la petición de enlace o BindRequest. El cliente no deberá lanzar operaciones entre dos peticiones de tipo enlace si forman parte de un enlace de tipo múltiple.

Un cliente debería abortar un enlace SASL enviando una petición de enlace (bindRequest) con un valor diferente en el campo de mecanismo (mechanism) de credenciales de SASL (SaslCredentials), o una opción de autentificación (AuthenticationChoice) distinta a SASL.

Si el cliente manda una petición de enlace con el campo mecanismo SASL vacío, el servidor deberá devolver una respuesta de enlace (BinDresponse) con el código de resultado establecido a authMethodNotSupported. Esto permitirá al cliente abortar la negociación si el cliente desea intentar de nuevo el mismo mecanismo SASL.

2.1.9.2 OPERACIONES LDAP: BINDRESPONSE

Consiste simplemente en una indicación desde el servidor del estado de la petición de autentificación del cliente.

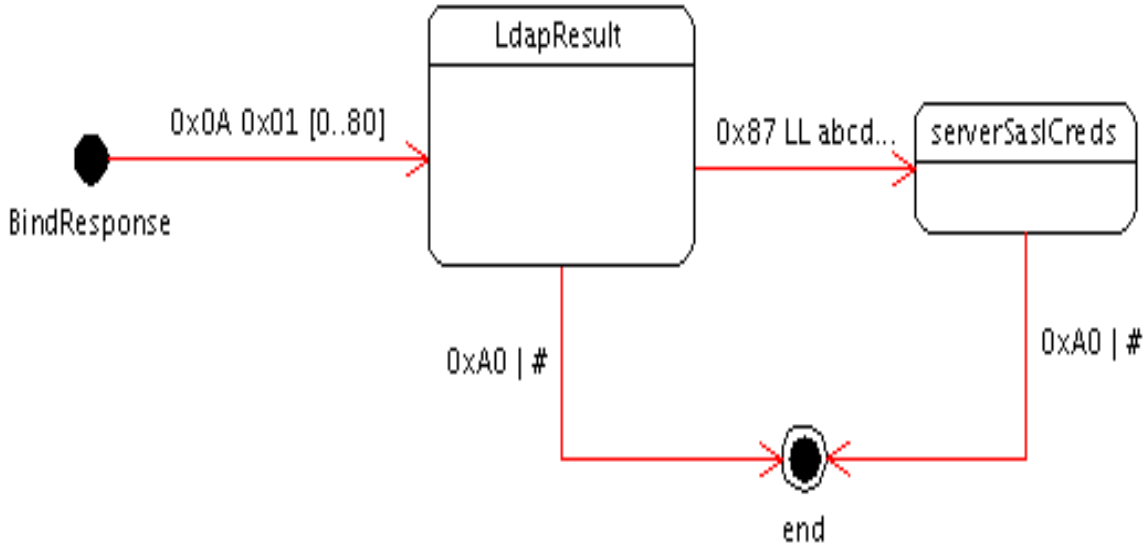

Figura 2.1-15: BindResponse

Una satisfactoria operación de enlace (Bind operation) está indicada mediante una respuesta de enlace (Binresponse) con un código de resultado (resultcode) definido como éxito.
En el resto de los casos, se definirá el código apropiado en la respuesta de enlace (BindResponse).

El código de resultado de error de protocolo (protocolError) podría indicar que el número de la versión proporcionado por el cliente no está soportado. En este caso, se asume que el servidor no soporta dicha versión de LDAP. El cliente que no tenga la posibilidad de continuar con otra versión del protocolo deberán dar por terminado la sesión LDAP.

El campo serverSaslCreds forma parte de un mecanismo de enlace de SASL para permitir al cliente autentificarse con el servidor que está comunicando, o para proporcionar una autentificación respuesta-desafío (challenge-response).

Si el cliente se conecta con la opción simple (simple choice), o el mecanismo SASL no requiere que el servidor retorne información al cliente, entonces este campo no debería estar incluido en la respuesta de enlace (BindResponse).

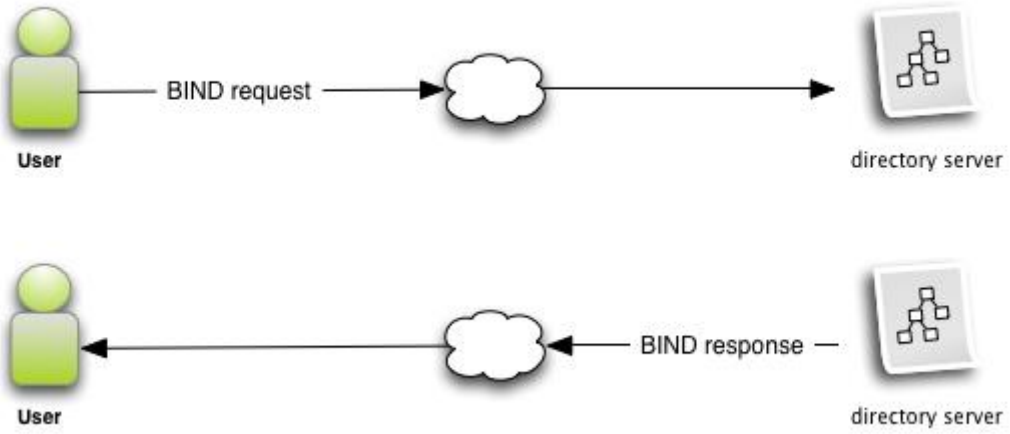

Figura 2.1-16: Comunicación BindRequest y BindResponse

2.1.9.3 OPERACIONES LDAP: UNBINDREQUEST

La función de la operación de desenlace es terminar una sesión LDAP. Debería ser tratada como la operación de salida.

El cliente, mediante la transmisión de la petición de desenlace (UnbindRequest) y el servidor, con su recepción, dan por terminada la sesión LDAP.

2.1.9.4 OPERACIONES LDAP: NOTIFICACIÓN SIN SOLICITUD

Una notificación no solicitada es un mensaje LDAP (LDAPMessage) enviado desde el servidor al cliente que no responde a ningún mensaje LDAP (LDAPMessage) recibido por el servidor. Es utilizado para señalar una condición extraordinaria en el servidor o en la sesión LDAP entre el cliente y el servidor. El servidor no espera respuesta alguna por el cliente.

Su estructura es un mensaje LDAP (LDAPMessage) con una identificación de mensaje (messageID) de cero y su operación de protocolo (protocolOp) establecida a respuesta extendida (extendedResponse). El campo de nombre de respuesta (responseName) de la respuesta extendida (extendedResponse) siempre contiene un LDAPOID que es único.

Tiene las siguientes especificaciones:
- El identificador objeto asignado en la notificación (especificado en el nombre de respuesta (responseName),
- El formato de los contenidos del valor de respuesta (responseValue),
- Las circunstancias que causarán la notificación ser enviada,
- La semántica del mensaje.

2.1.9.5 OPERACIONES LDAP: NOTICIA DE DESCONEXIÓN

La notificación podría ser usada por el servidor para advertir al cliente que el servidor está a punto de terminar la sesión LDAP. Esta notificación permite asistir a los clientes a distinguir entre una condición excepcional del servidor o un fallo de red. Esta notificación no es una respuesta a una petición de desenlace (UnBindRequest) por el cliente.

El nombre de respuesta (responseName) es "1.3.6.1.4.1.1466.20036", el valor de respuesta (responseValue) es nulo, y el código de respuesta es utilizado para indicar la razón de la desconexión. Cuando el código de resultado (resultCode) de requerimiento de autentificación fuerte (strongerAuthRequired) es devuelto en este mensaje, indica que el servidor ha detectado que una asociación de seguridad establecida entre el cliente y el servidor ha fallado inesperadamente.

Con la transmisión de esta noticia de desconexión, el servidor termina su sesión LDAP.

2.1.9.6 OPERACIONES LDAP: SEARCH OPERATION

Es utilizada para pedir al servidor que devuelva, al sujeto para acceder a los controles (controls) y otras restricciones, una serie de entradas que cumplan unos complejos criterios de búsqueda. Esto puede utilizarse para leer atributos de una entrada simple, desde entradas inmediatamente subordinadas a una entrada particular, o desde un completo subárbol de entradas.

Al igual que una petición de enlace para autentificación, cuando se realiza una búsqueda el primer paso es la petición por parte del cliente al servidor donde se quiere implementar dicha búsqueda. Constituida por lo siguiente:

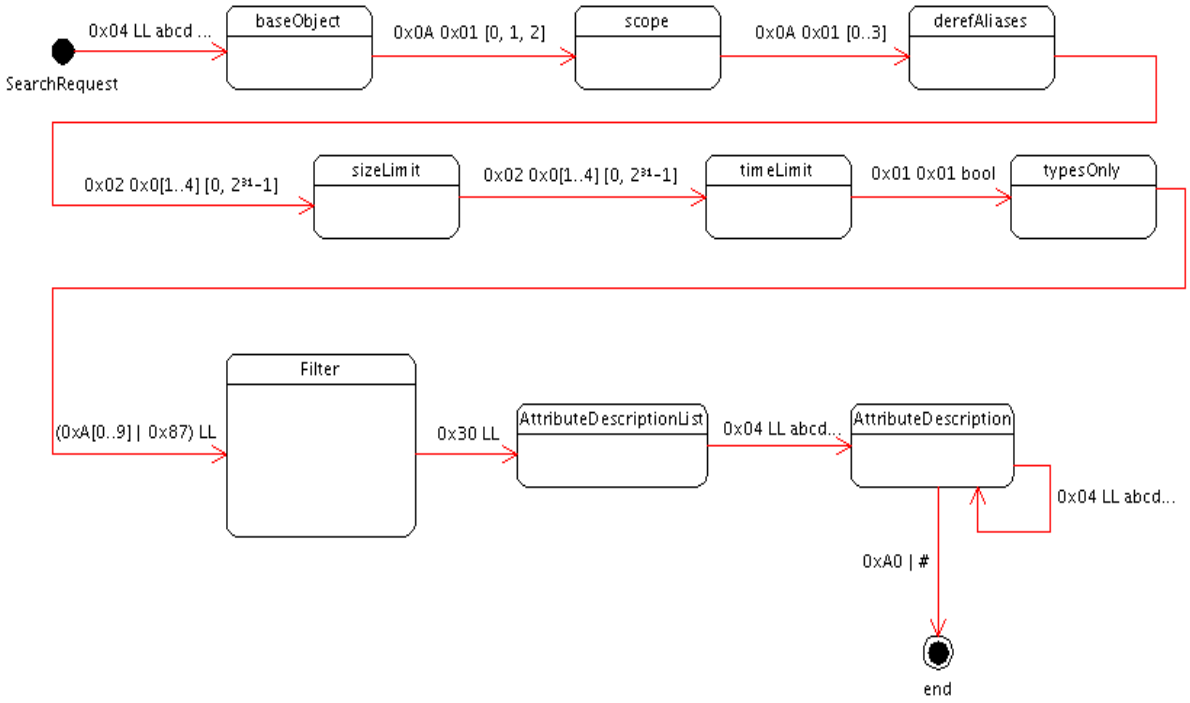

Figura 2.1-17: SearchRequest

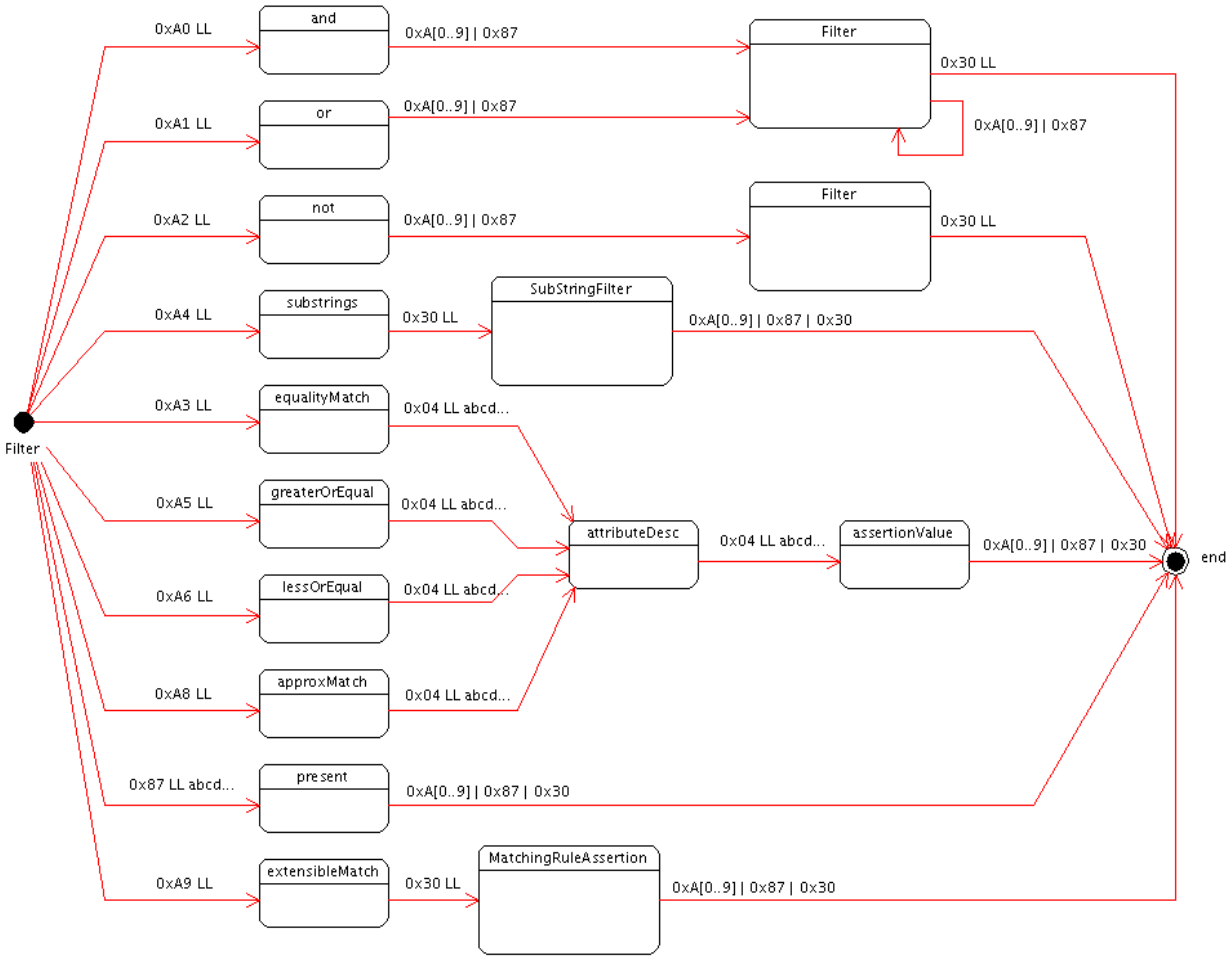

Figura 2.1-18: Filter

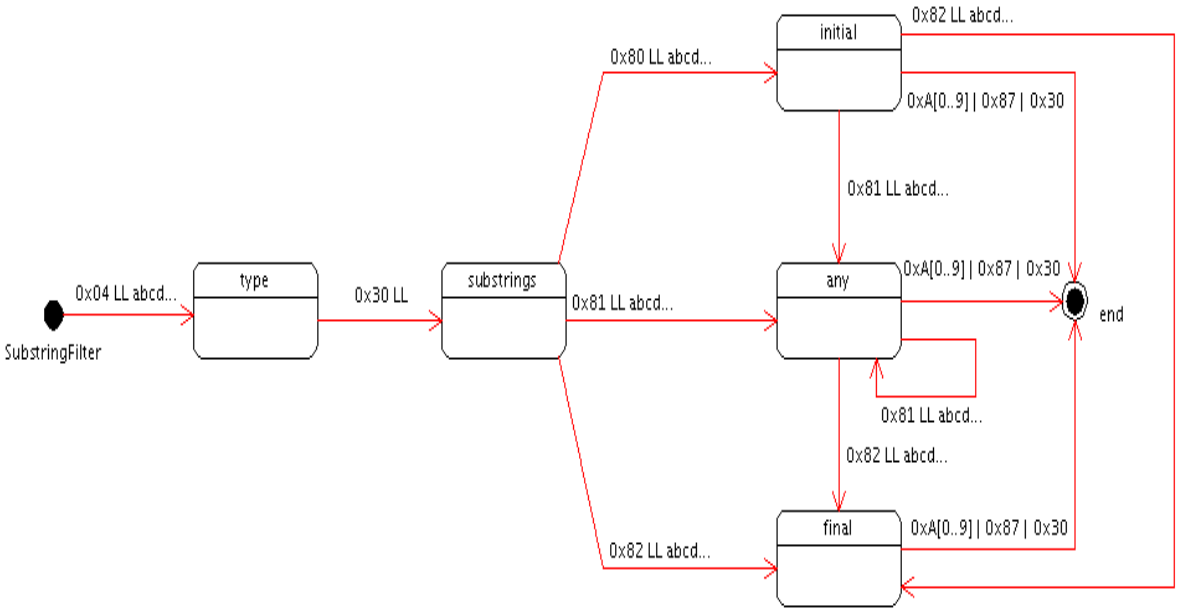

Figura 2.1-19: SubStringFilter

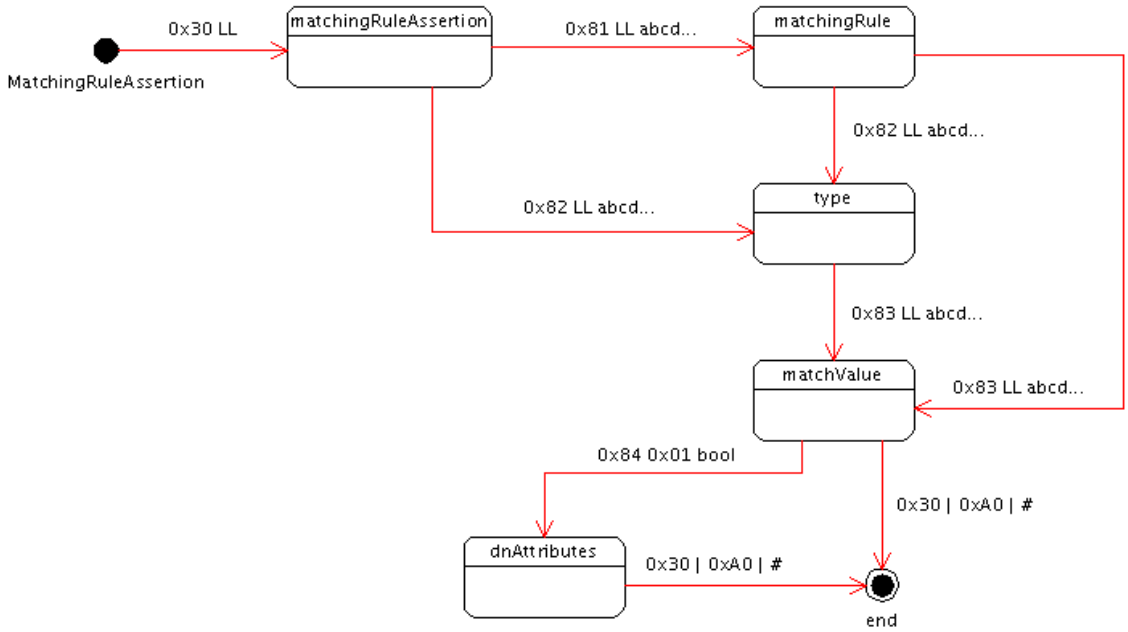

Figura 2.1-20: MatchingRuleAssertion

- BASEOBJECT: el nombre de la entrada del objeto base relativo a la búsqueda.
- SCOPE: el alcance de la búsqueda que se va a realizar. Contiene lo siguiente:
  - baseObject: el alcance se limita a la entrada denominada por el objeto base.
  - singleLevel: el alcance se limita a los subordinados inmediatos de la entrada denominada por el objeto base.
  - wholeSubtree: el alcance se limita a la entrada denominada por el objeto base y todo sus subordinados.

- DEREFALIASES: es un indicador de si los alias de entradas o no son referenciados durante las etapas de la operación de búsqueda. La semántica incluye la siguiente información:
  - neverDerefAliases: no existen aliases desreferenciados en la búsqueda o en la localización del objeto base de la búsqueda.
  - derefInSearching: desreferenciar cualquier alias sin el alcance de búsqueda mientras se busca subordinados en el objeto base.
  - derefFindingBaseObj: desreferenciar los alias cuando se localiza el objeto base en la búsqueda, pero no cuando se busca subordinados del objeto base.
  - derefAlaways: deresferenciar ambos alias en la búsqueda y en la localización del objeto base de la búsqueda.

- SIZELIMIT: el límite de tamaño que restringe el máximo número de enetradas que se han devuelto como resultado de la búsqueda.

• TIMELIMIT: el límite de tiempo que restringe el máximo tiempo (en segundos) permitido por una búsqueda.

• TYPESONLY: un indicador de si la búsqueda contiene tanto descripción de atributos y valores, o sólo descripción de atributos.

• FILTER: un filtro que define condiciones que deberán ser cumplidas para que la búsqueda devuelva un valor para una entrada dada.

• ATTRIBUTES: una lista de selección de los atributos que se devuelven desde cada entrada que cumple el filtro de búsqueda.

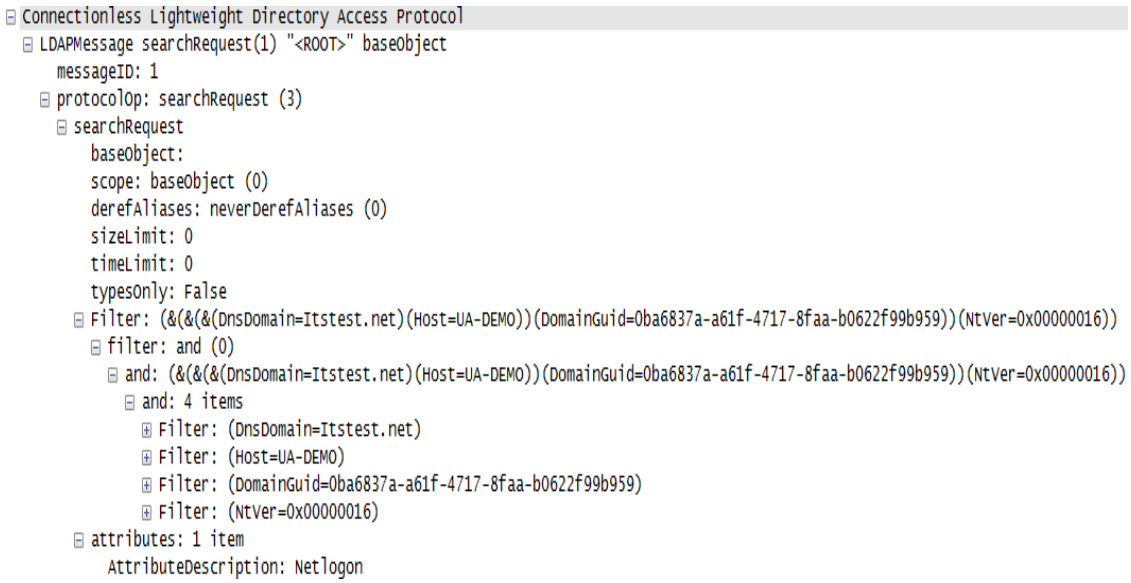

Figura 2.1-21: Trama SearchRequest

2.1.9.7 OPERACIONES LDAP: SEARCH RESULT

Los resultados de la operación de búsqueda son devueltos como cero o más entradas de resultado de búsqueda (SearchResultEntry) y/o mensajes de referencia de resultado de búsqueda (SearchResultReference), seguidos por un único mensaje de SearchResultDone.

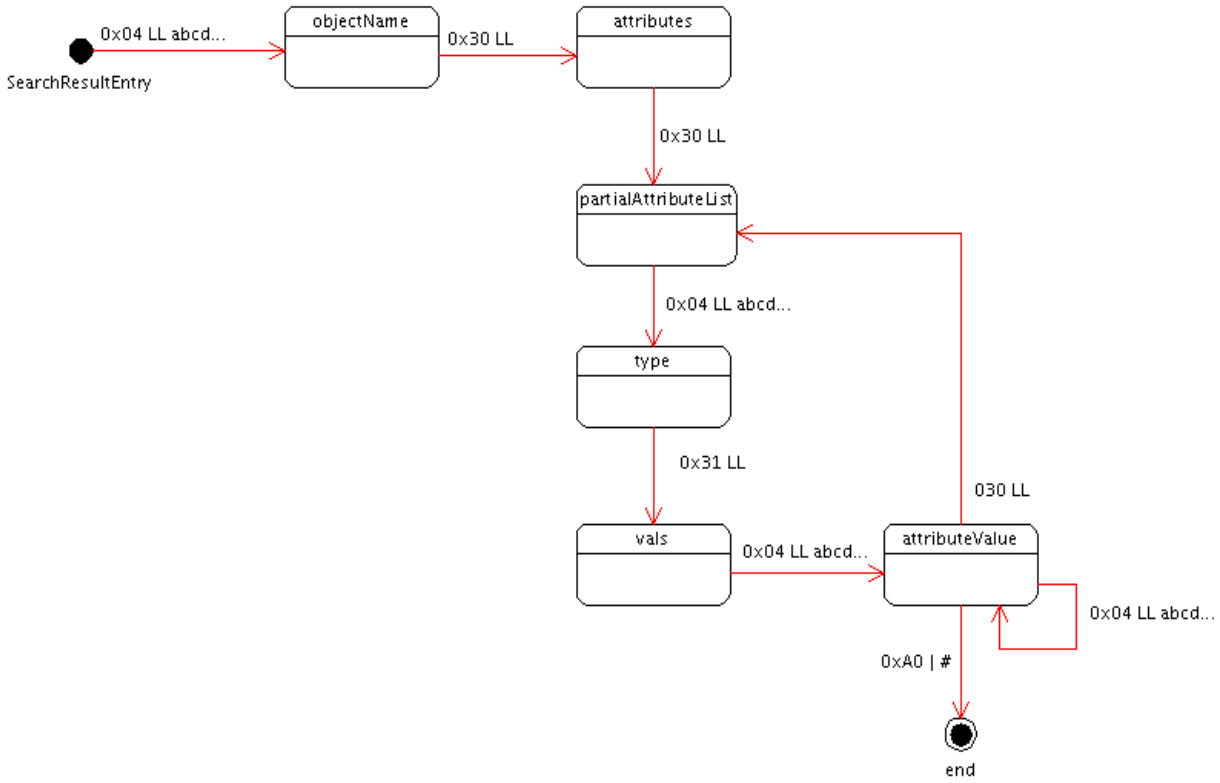

Figura 2.1-22: SearchResultEntry

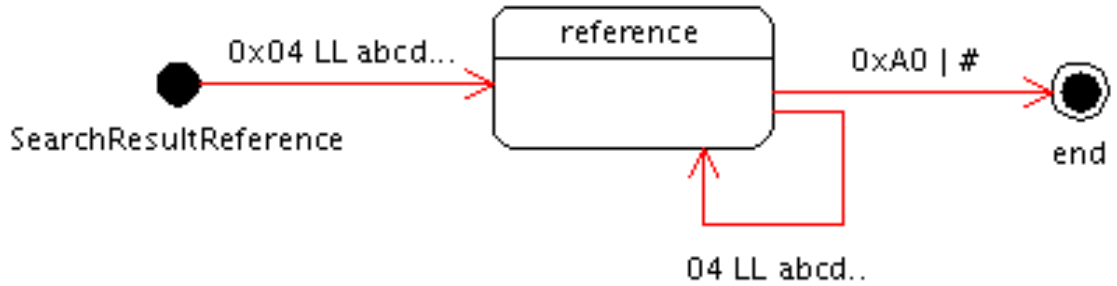

Figura 2.1-23: SearchResultReference

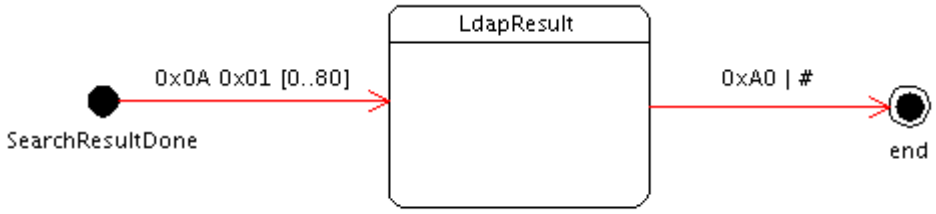

Figura 2.1-24: SearchResultDonde

Cada entrada de resultado de búsqueda (SearchResultEntry) representa una entrada encontrada durante la búsqueda. Cada referencia de resultado de búsqueda (SearchResultReference) representa un área no explorada todavía durante la búsqueda.

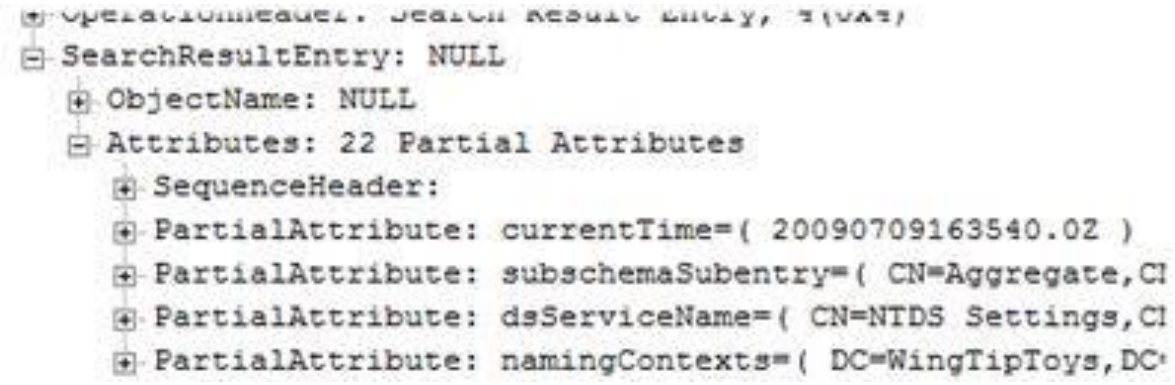

Figura 2.1-25: Trama SearchResultEntry

Ambos mensajes pueden volver en cualquier orden. Tras seguir todas las respuestas de SearchresultReference y SearchResultEntry, el servidor devuelve una respuesta SearchResultDone, que contiene un indicador de éxito o los detalles de los errores que han ocurrido.
Si el servidor pudo localizar la entrada referenciada por el baseObject pero no pudo buscar una o más entradas no locales, el servidor debería devolver uno o más mensajes de SearchResultReference, cada uno conteniendo la referencia a otro set de servidores para continuar la operación.

Un servidor no deberá devolver ningún mensaje SearchResultReference si no ha localizado el baseObject y si no ha encontrado ninguna entrada. En este caso, debería devolver un SearchResultDonde conteniendo la referencia o el código de resultado noSuchObject.

Si el cliente quiere continuar con la búsqueda, deberá crear una nueva operación de búsqueda para cada SearchResultReference que se ha devuelto.

Los clientes que continúan la búsqueda de referencias deberán asegurarse de no repetirse entre servidores. No deberán repetir contactos al mismo servidor para la misma petición con los mismos parámetros.

El cliente deberá abandonar individualmente las secuencias de las operaciones de búsquedas que desee mediante la operación de abandono (Abandon operation).

Los valores de SearchResultReference que se rigen por las URLs de LDAP siguen las siguientes normas:

- La parte <dn> de una URL LDAP deberá estar presente con el nuevo destino del nombre objeto.

- Si la parte <filter> de una URL de LDAP está presente, el cliente utilizará este filtro en la siguiente petición de progreso en la búsqueda, y si no está presente el cliente utilizará el mismo filtro que ha utilizado en esa búsqueda.

- Si el alcance de búsqueda (search scope) fue singleLevel, la parte <scope> de la URL LDAP será "base".

- Se recomienda que la parte <scope> esté presente para evitar ambigüedades.

2.1.9.8 OPERACIONES LDAP: MODIFY OPERATION

La operación de modificación permite al cliente la petición de modificación de una entrada que tiene que ser ejecutada por el servidor.

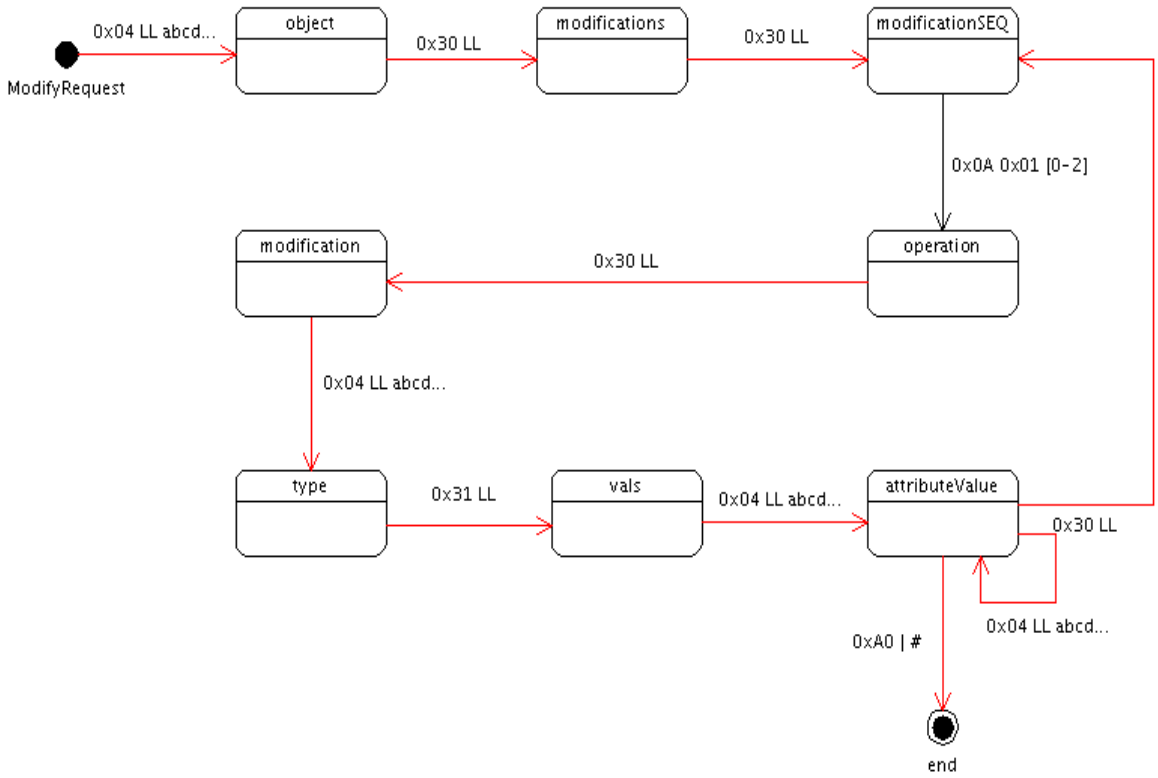

Figura 2.1-26: ModifyRequest

Los campos son los siguientes:

- Object: el valor de este campo contiene el nombre de la entrada que va a ser modificada.
- Changes: una lista de modificaciones a realizarse sobre una entrada. La lista entera deberá ser realizada en el orden que han sido listadas como una única operación atómica.
- Operation: especifica el tipo de modificación que va a ser implementada.
  - o Add: añade valores al atributo de modificación.
  - o Delete: elimina el valor listado del atributo de modificación.
  - o Replace: substituye todos los valores existentes del atributo de modificación con los nuevos valores listados.

- Modification: un PartialAttribute utilizado para alojar el tipo de atributo y/o los valores que están siendo modificados.

2.1.9.9 OPERACIONES LDAP: MODIFYRESPONSE

Tras recibir una petición de modificación, el servidor intenta realizar las modificaciones necesarias al directorio y devuelve el resultado en la respuesta de modificación, definida de la siguiente manera:

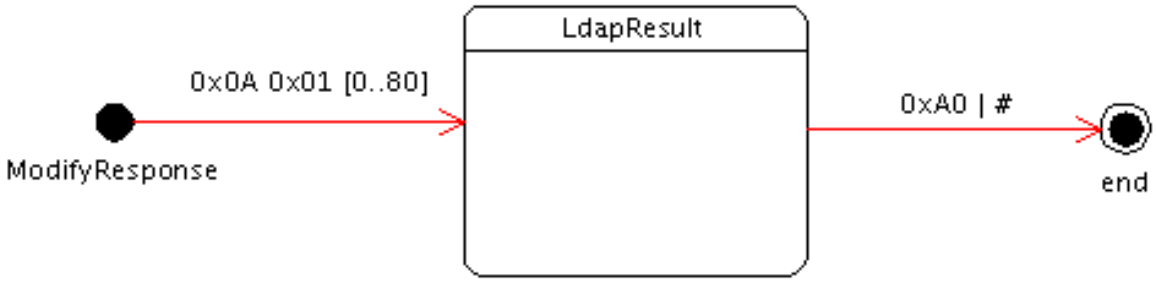

Figura 2.1-27: ModifyResponse

El servidor devolverá al cliente una única respuesta de modificación indicando tanto la correcta finalización de la modificación del directorio, o las razones por las que dicha modificación ha fallado. Si no se recibe la respuesta de modificación, el cliente no puede determinar si la modificación fue aplicada.

2.1.9.10 OPERACIONES LDAP: ADD OPERATION

La operación de adición permite al cliente realizar una petición de adición con la entrada dentro del directorio. Es definida de esta manera:

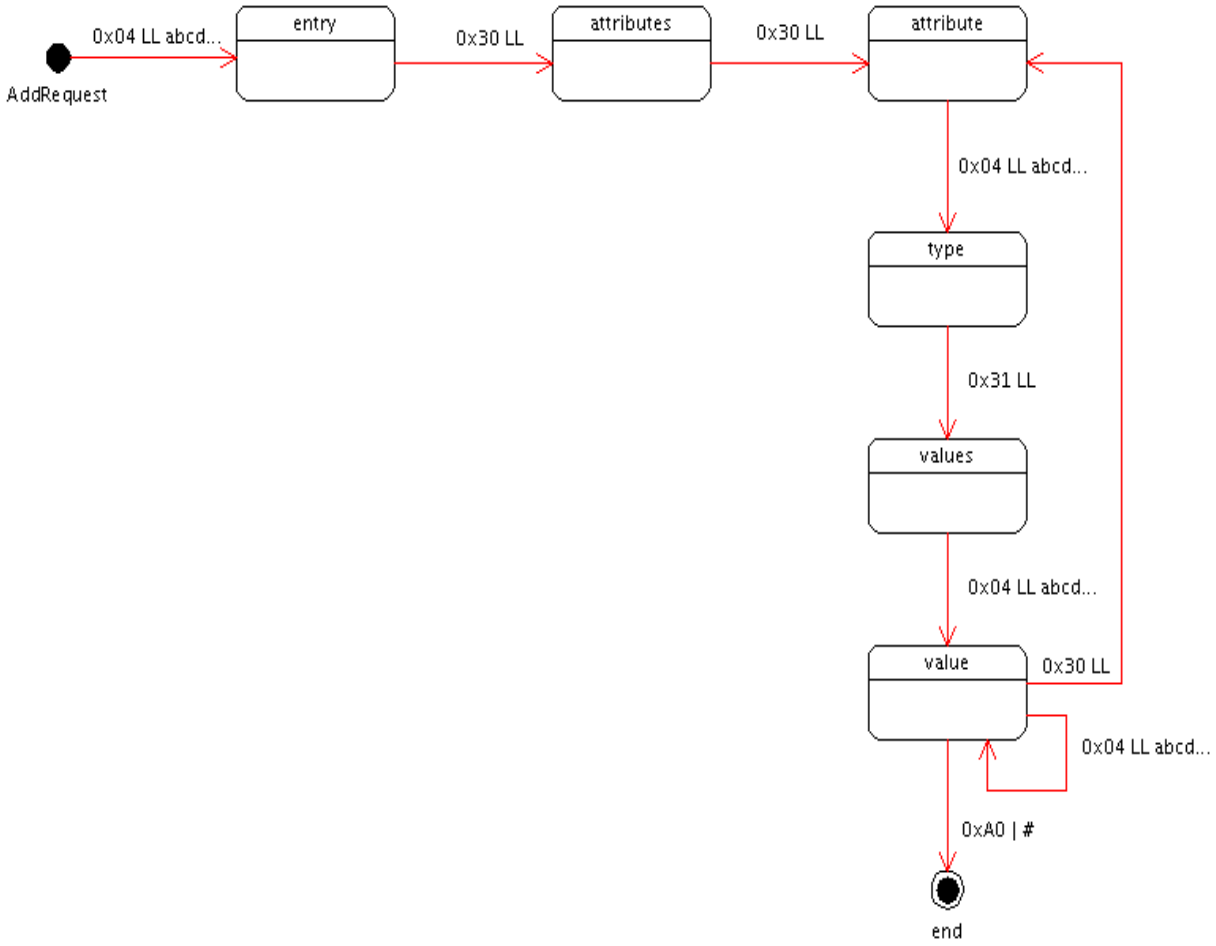

Figura 2.1-28: AddRequest

- Entry: el nombre de la entrada que va a ser añadida.

- Attributes: la lista de los atributos que, componen el contenido de la entrada a añadir.

Tras recibir una petición de adición, el servidor intentará acceder a la entrada requerida. El resultado de ese acceso se traducirá en una respuesta de adición, definida de la siguiente manera:

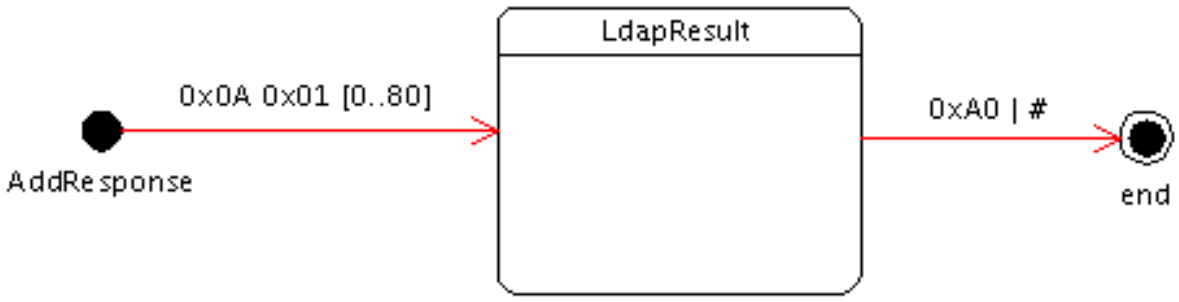

Figura 2.1-29: AddResponse

Una respuesta de éxito indica que una nueva entrada ha sido añadida al directorio.

2.1.9.11 OPERACIONES LDAP: DELREQUEST

Permite al cliente requerir la eliminación de la entrada del directorio.

Figura 2.1-30: DelRequest

La petición de eliminación consiste en el nombre de la entrada a ser eliminada. Sólo la entradas que no tienen subordinadas pueden ser eliminadas por esta operación.

Después de la petición de borrado, el servidor intentará borrar la entrada indicada y devolverá el resultado en la respuesta de borrado (Delte Response) definida como:

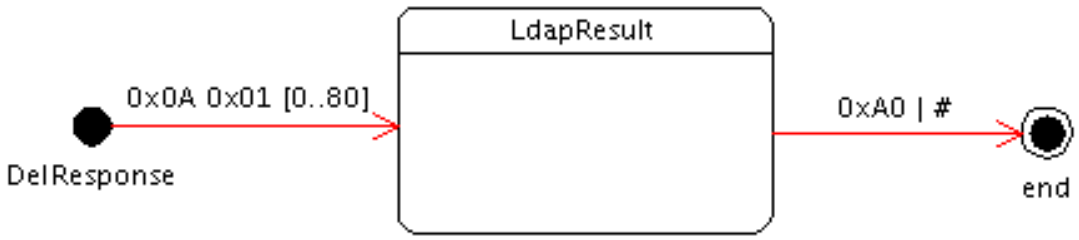

Figura 2.1-31: DelResponse

2.1.9.12 OPERACIONES LDAP: MODIFY DN OPERATION

Permite al cliente cambiar el nombre de distinción relativo (RDN) de una entrada en el directorio y/o mover el subárbol de entradas a una nueva localización en dicho directorio.

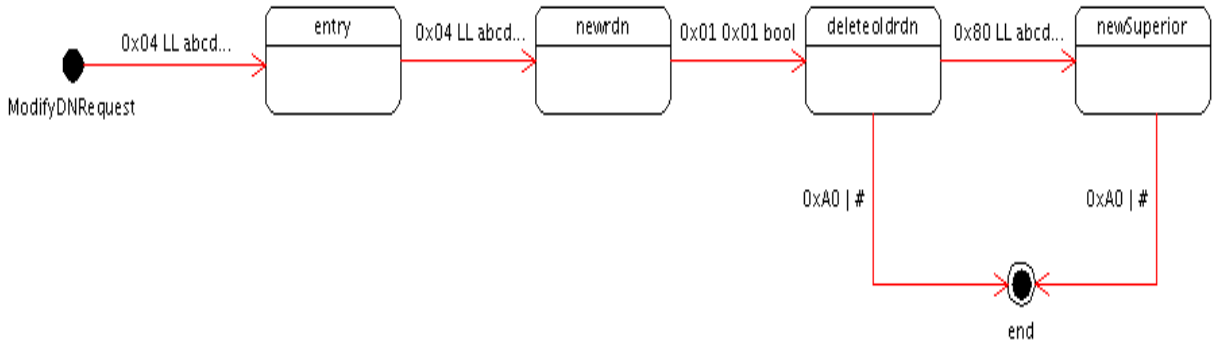

Figura 2.1-32: ModifyDNRequest

- Entry: el nombre de la entrada a ser eliminada. Esta entrada podrá o no, tener entradas subordinadas.

- Newrdn: el nuevo RDN de una entrada. El valor del anterior RDN se suple moviendo la entrada a una nueva superior sin cambiar dicho RDN.

- Deleteoldrdn: un campo boolean que controla si el valor de atributo del anterior RDN es retenido como atributo de entrada o eliminado de dicha entrada.

- newSuperior: si está presenta, es el nombre de una entrada de objeto existente que pasa a formar parte del inmediato superior de la existente entrada.

El servidor intentará cambiar el nombre y, devolverá el resultado en la respuesta de modificación de DN (ModifyDNResponse):

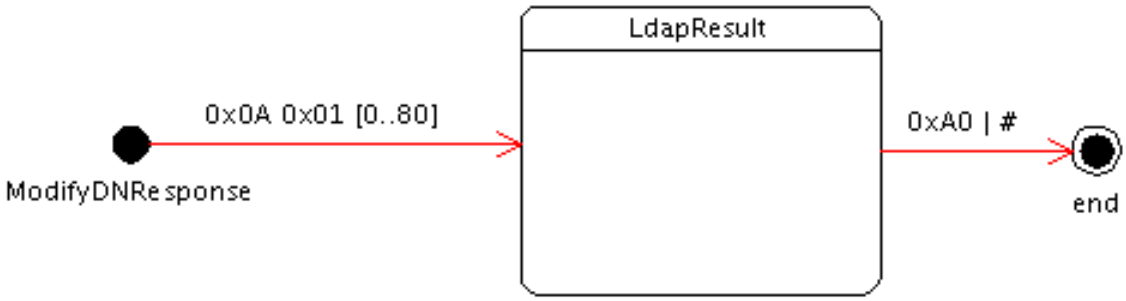

Figura 2.1-33: ModifyDNResponse

2.1.9.13 OPERACIONES LDAP: COMPAREREQUEST

Permite al cliente comparar una serie de valores con los valores de atributo particular en una determinada entrada del directorio.

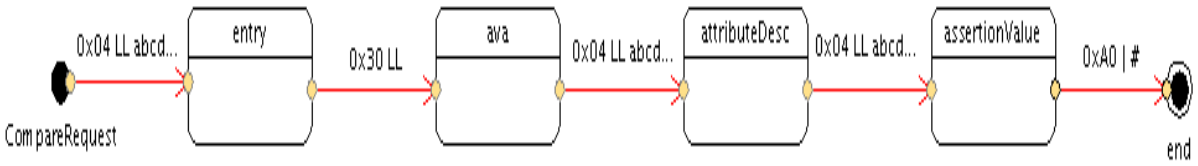

Figura 2.1-34: CompareRequest

- entry: el nombre de la entrada a comparar.
- Ava: el valor de los atributos que son comparados.

La respuesta de comparación se define de la siguiente manera:

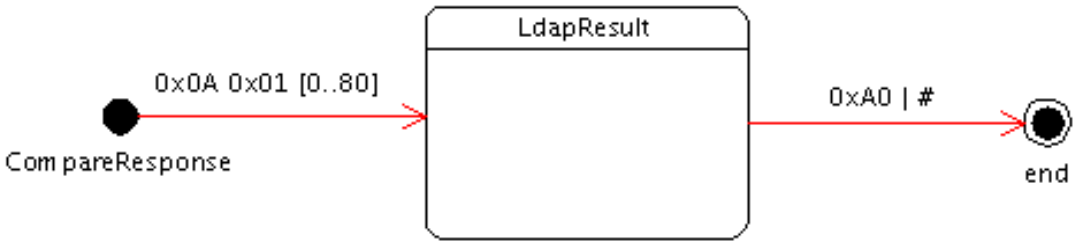

Figura 2.1-35: CompareResponse

El código de resultado está establecido a compareTrue, compareFalse, o un error apropiado. CompareTrue indica que un valor o subtipo cumple la serie de valores que se especifican en el campo ava. compareFalse está establecido cuando no se cumplen dichos valores especificados. Otros códigos de resultado incidan que la comparación fue indefinida, o algún error en dicha comparación.

2.1.9.14 OPERACIONES LDAP: ABANDONREQUEST

Permite al cliente realizar una petición para que el servidor abandone una operación incompleta.

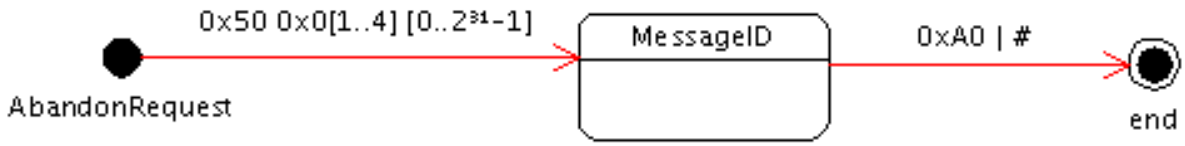

Figura 2.1-36: AbandonRequest

El MessageID es distinto de otro MessageID de una operación de abandono previa y, es el mismo de la capa del mensaje LDAP que se ha pedido previamente.
No hay respuesta definida para la petición de abandono, el servidor debería abandonar la operación identificada por el MessageID.

2.1.9.15 OPERACIONES LDAP: EXTENDEDREQUEST

Permite operaciones adicionales para servicios que no son todavía disponibles en el protocolo, por ejemplo, añadir operaciones para instalar una capa de seguridad de transporte.

Permite a los clientes realizar peticiones y recibir respuestas con sintaxis y semántica predefinida.

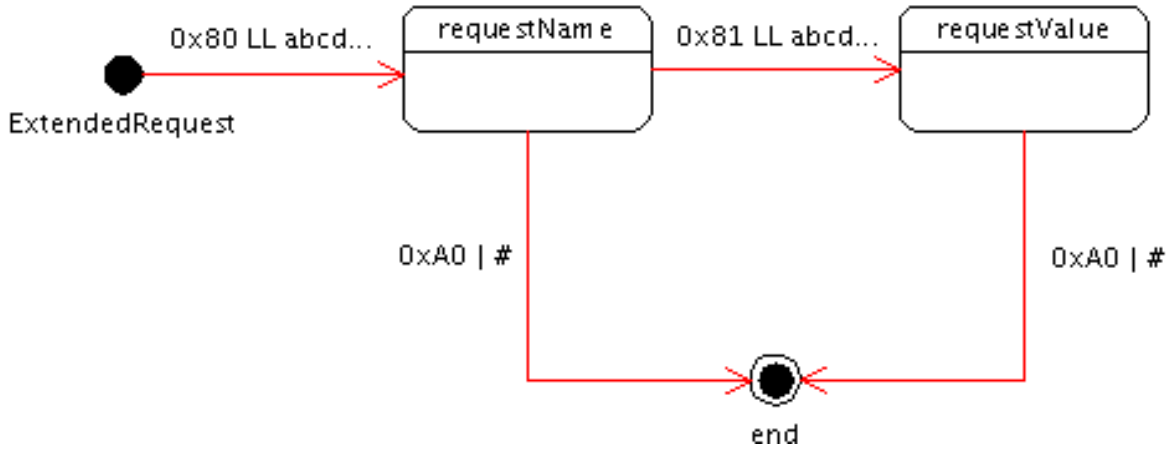

Figura 2.1-37: ExtendedRequest

- requestName: es una representación del único identificador objeto correspondiente a la petición.

- requestValue: información de un formulario definida por la petición.

El servidor responderá con un LDAPMessage mediante un ExtendedResponse:

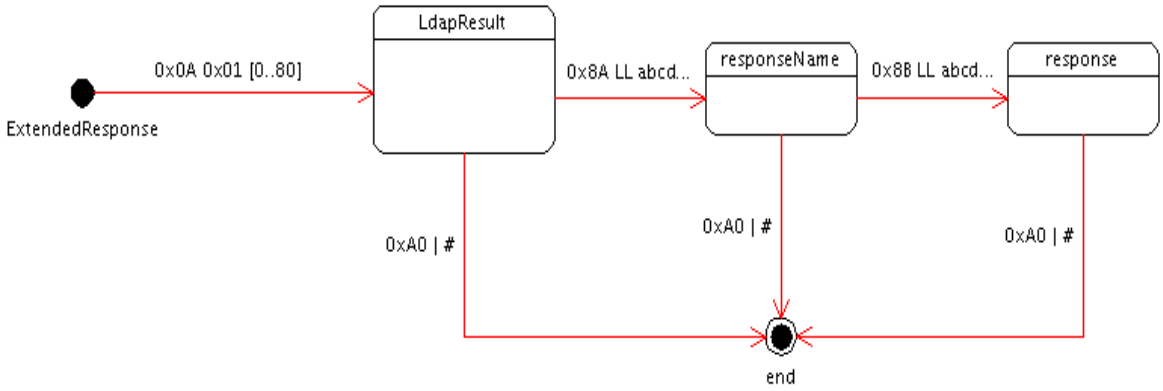

Figura 2.1-38: ExtendedResponse

• ResponseName: contiene el LDAPOID que es único para esta operación extendida o respuesta.

• Responsevalue: contiene información asociada con la operación. El formato de este campo y del campo requestValue está especificado por la operación extendida.

2.1.10 MENSAJE INTERMEDIATERESPONSE

Mientras la operación de búsqueda proporciona un mecanismo para devolver múltiples mensajes de respuesta para una única petición de búsqueda, otras operaciones, por naturaleza, no proporcionan múltiples mensajes de respuesta.

El mensaje IntermediateResponse proporciona un mecanismo general para estas operaciones definidas como única-petición/múltiples-respuestas o proporciona un control por si se requiriera extender las existentes operaciones LDAP y, con ello devolver información de respuesta Intermediate.

Los mensajes IntermediateResponse no deberían ser devueltos al cliente hasta que el cliente realice una petición específica con dicha devolución.

2.1.11 OPERACIÓN STARTTLS

La operación de capa de inicialización de transporte de seguridad (StartTLS) inicializa la instalación de una capa TLS.

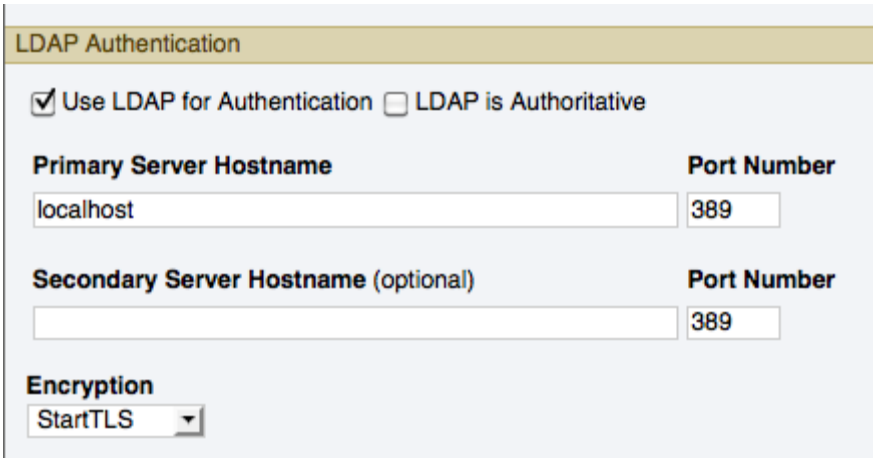

Figura 2.1-39: LDAP STARTTLS Authentication

2.1.11.1 OPERACIÓN STARTTLS: PETICIÓN

Un cliente solicita el establecimiento TLS transmitiendo un mensaje de petición StartTLS al servidor. Está definido dicho mensaje como un ExtendedRequest. El requestName es "1.3.6.1.4.1.1466.20037", y el requestValue está siempre vacío.

El cliente no deberá enviar ninguna PDU LDAP en esta capa de mensaje LDAP tras la petición hasta que se reciba una respuesta y, en el caso de que sea correcta, complete la negociación TLS.

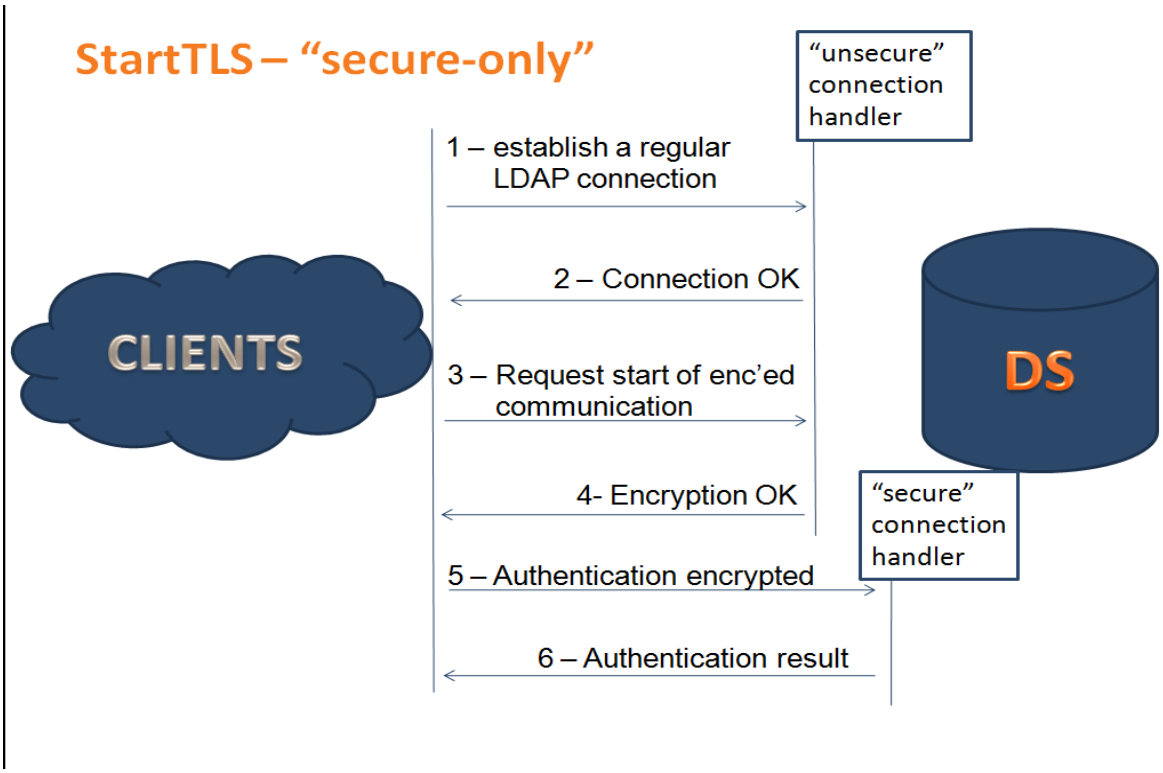

Figura 2.1-40: Mecanismo de seguridad STARTLS

2.1.11.2 OPERACIÓN STARTTLS: RESPUESTA

Cuando se recibe una petición StartTLS, los servidores que soportan la operación deberán devolver un mensaje de respuesta StartTls al demandante. El responseName es ".1.3.6.1.4.1.1466.20037". El responseValue es siempre vacío.

2.1.11.3 OPERACIÓN STARTTLS: BORRADO

El cliente o el servidor deberán borrar la capa TLS y dejar la capa del mensaje LDAP intacta enviando y recibiendo la alerta de cierre TLS. Se deberá esperar a recibir la alerta correspondiente a ese mensaje antes de enviar PDUs LDAP de otros mensajes.

2.1.12 CODIFICACIÓN DEL PROTOCOLO: CONEXIÓN Y TRANSFERENCIA

Este protocolo está diseñado para funcionar sobre sistemas orientados a conexión, fiables, donde el dato transmitido está dividido en octetos (8-bit units), donde cada octeto y cada bit son significativos.

Este servicio es generalmente desarrollado en aplicaciones que proporcionan o utilizan servicios de directorios basados en x.500 sobre Internet. Esta especificación fue generalmente implementada sobre TCP.

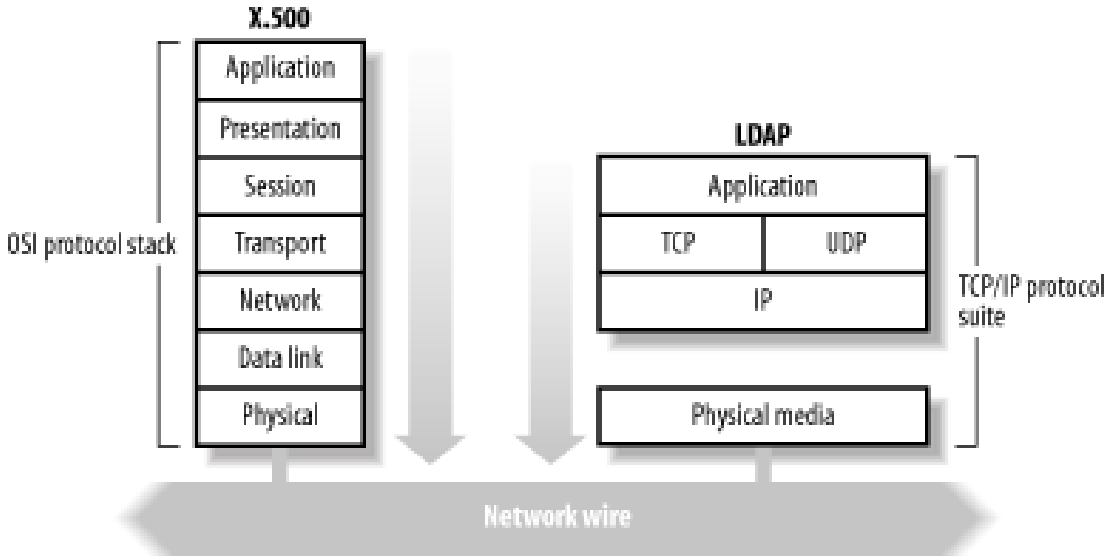

Figura 2.1-41: Capas de transporte LDAP

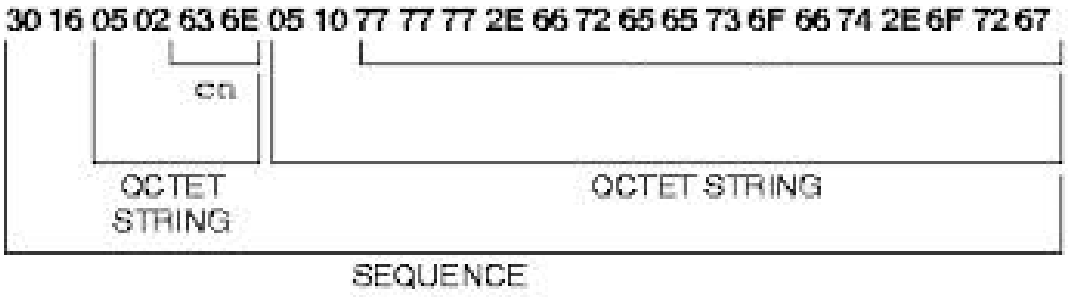

Figura 2.1-42: Codificación BER

Los elementos del protocolo LDAP deberían ser codificados utilizando las reglas de codificaciones básicas (BER) con las siguientes restricciones:

- Sólo la forma de la longitud de codificación definida es utilizada.

- Los valores de OCTET STRING son codificados en la forma primitiva únicamente.

- Si el valor de tipo BOOLEAN es verdadero, la codificación del valor octeto será establecida a “FF”.

- Sólo algunos tipos de BOOLEAN e INTEGER tienen valores por defecto en esta definición de protocolo, el resto es valor vacío.

La codificación de las PDU del LDAPMessage están mapeadas directamente sobre el stream de bytes de TCP utilizando la codificación BER anteriormente descrita.

Es recomendable que el servidor escuche por el puerto 389 (Internet Assigned Numbers Authority (IANA) – assigned LDAP Port). Aunque también se deberá poder proporcionar escuchar por otro puerto diferente. Los clientes deberán soportar contactar con el servidor sobre cualquier puerto TCP.

2.1.13 FINALIZACIÓN DE LA SESIÓN LDAP

Es típicamente inicializada por el cliente enviando un UnbindRequest, o mediante el servidor enviando una noticia de desconexión (Notice of Disconnection). En estos casos, se terminará la sesión cesando el intercambio de la capa de los mensajes LDAP, finalizando cualquiera capa SASL, cualquier capa TLS y, cerrando la conexión del transporte.

Se deberá avisar que cualquier comunicación sería peligrosa, y que se debería terminar la sesión inmediatamente cesando dicha comunicación y cerrando la conexión transporte.

2.1.14 CONSIDERACIONES DE SEGURIDAD

Esta versión del protocolo proporciona facilidad para la autentificación simple utilizando una contraseña de texto plano, como cualquier mecanismo SASL. Instalando capas SASL y/o TLS se puede proporcionar servicios de integridad y de seguridad.

A continuación se ilustra un adaptador TLS que permite una mayor robustez frente a un ataque malicioso tipo Man In the Midle ocasionado por una tercera persona ajena al sistema.

La respuesta se proporciona por el túnel SSL evitando que el hacker pueda entrar en el canal de seguridad y adquirir cualquier tipo de información valiosa. Los certificados identifican a la persona original otorgando una clave privada única.

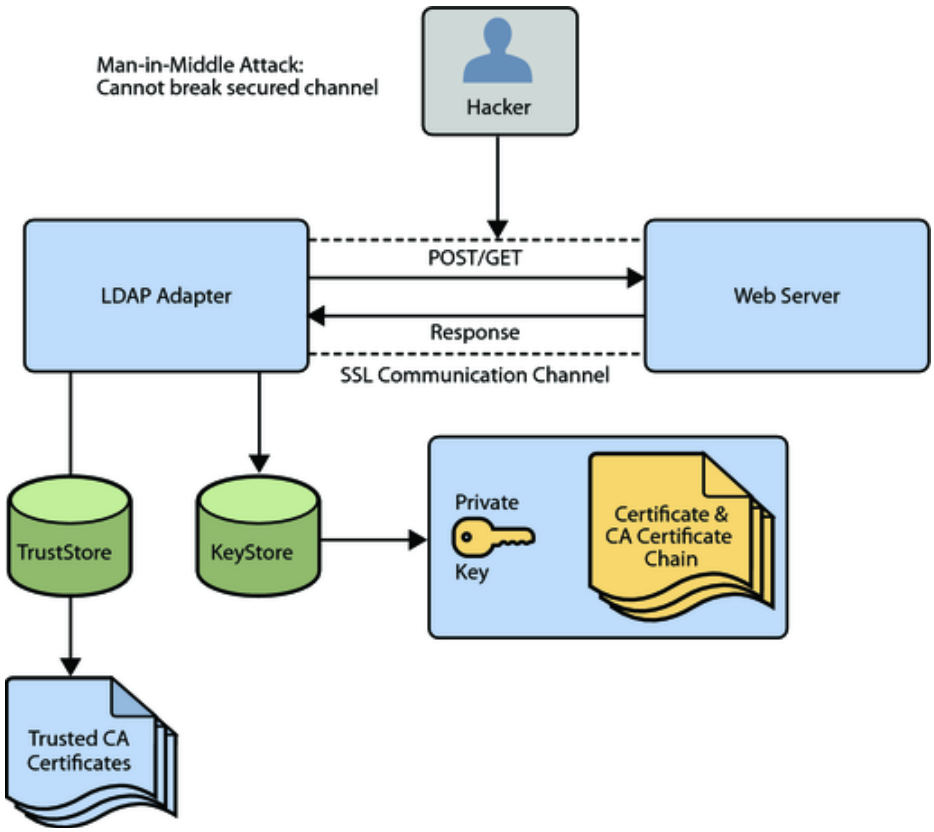

Figura 2.1-43: SSL Security Channel

También está permitido que el servidor pueda devolver sus credenciales al cliente. La utilización de contraseña de texto plano no se recomienda cuando el servicio de transporte no puede garantizar confidencialidad y puede ocasionar en la revelación de la contraseña por partes no autorizadas.

Los servidores deberán prevenir modificaciones de los directorios por clientes que se autentifiquen de manera anónima.

Los intercambios de autentificación SASL no proporcionan datos confidenciales o protección de integridad para los campos de versión o nombre de BindRequest o para los campos tipo resultCode, diagnosticMessage o referral de BindResponse, ni ninguna información contenida en controles asociados a peticiones o respuestas de tipo enlace (Bind). Por tanto, esta información contenida en estos campos no deberá ser relevada si no está protegida de otra manera.

Los factores de seguridad podrían cambiar durante el transcurso de una sesión LDAP o en el desarrollo de una operación particular. La implementación deberá ser robusta para manejar cualquier cambio de este tipo. Pese a estos cambios, sería apropiado que la sesión continuara con la operativa. La situación variará con el cambio de seguridad y se deberá continuar o no en función de dicho cambio.

Si se percibe que se produce un ataque que cambiaría la naturaleza del protocolo y causaría daño a la comunicación en cualquier capa de la sesión LDAP, se deberá terminar inmediatamente la sesión LDAP.

2.2 FALLOS DE COMUNICACIÓN

Una vez descrito con gran detalle el funcionamiento del protocolo LDAP, se muestran a continuación una serie de ejemplos con fallos en la transmisión de dicho protocolo ya que en este proyecto el objetivo es diseñar un programa que analice su comportamiento e intente encontrar las causas y proponer las soluciones ante estos fallos.

Los árboles LDAP han sido utilizados en muchas organizaciones y los posibles ataques que se han descubierto, implementado o ejecutado siempre se han visto restringidos a entornos muy concretos.

Como se ha explicado previamente, el sistema de transmisión LDAP en una comunicación debe seguir una serie de normas y criterios. Estas reglas se deben cumplir tanto en el lado del cliente como en el servidor con el fin de asegurar la comunicación y potenciar su fiabilidad.

Por tanto, el primer problema en la comunicación que se debería analizar es que cualquiera de los campos relativos a las normas principales a seguir se están devolviendo de manera adecuada. El disector muestra el campo que el usuario que está utilizándolo quiere obtener y, comparando su valor con lo esperado, es capaz de entender el motivo del fallo en la comunicación si ese fallo se ha detectado.

Más adelante, se entrará en detalle sobre cómo utilizar el programa y, se mostrarán ejemplos de salidas que devolvería el disector. Ahora se muestra una lista de los fallos de comunicación típicos y, que se considerarán los **fallos generales de autenticación internos**.

- Error de conexión
- Tiempo de espera de conexión
- Error de sintaxis de filtro
- Error de atributo de búsqueda
- Error de comunicación
- Error de falta de recursos

Este tipo de fallos son relativos al intento de autenticación del cliente y, deben ser detectados al inicio de la conexión. Una manera muy sencilla de detectar un fallo de este tipo es, analizando que los campos devueltos por la petición de enlace con el número de identificación de mensaje son similares a la petición de respuesta. También observando el código de resultado de la operación de enlace se detectaría un posible fallo.

En estos casos, una aplicación mostraría el siguiente mensaje:

Su nombre de usuario o la contraseña no son válidos.

Figura 2.2-1: Fallo de autenticación

Y, podría mostrar un mensaje que apareciera en el registro:

CWZIP4665W: La conexión a LDAP ha fallado. Se ha producido el error siguiente: CommunicationException: 172.16.248.10:389

Figura 2.2-2: CommunicationException

Una vez se detectara el fallo y, si fuese posible reparar la conexión, se informaría que todo ha funcionado correctamente.

CWZIP4666I: La conexión a LDAP se ha restaurado.

Figura 2.2-3: Conexión LDAP restaurada

Otros fallos que serían posibles en una comunicación LDAP podrían darse **durante la comunicación**. No obstante, serían fallos cuyos campos tendrían una descripción explicada en detalle en la sección relativa a la descripción del protocolo.

El campo para estos fallos es más amplio ya que un fallo en la comunicación puede darse en ambos lados tanto en el cliente como en el servidor y, el motivo del fallo no puede acotarse previamente. Son muchos los motivos que pueden ocasionar un cierre en la comunicación LDAP pero el disector es capaz de extraerlos y compararlos. De esta manera se solucionarían.

Por ejemplo, si durante una comunicación LDAP el servidor no es capaz de devolver el número de identificación de mensaje relativo al mensaje que el cliente ha solicitado en ese instante porque ha respondido a otro mensaje posterior debido a que el servidor está saturado, la transmisión del protocolo no sería adecuada y la sesión se terminaría.

El disector analizaría el campo messageID y vería que en una petición BindRequest con messageID “0x01”, se ha devuelto una respuesta BindResponse con messageID “0x02”. El usuario detectaría de manera rápida y sencilla y, podría solucionar el problema.

No sólo hay fallos **generales de autenticación internos o** fallos **durante la comunicación,** también se pueden producir fallos en la **seguridad del protocolo** y son los más importantes y que más dinero supondrían a una empresa.

La proliferación de los sistemas de directorio como repositorio central de información, apoyados en árboles LDAP, de los cuales son algunos ejemplos destacados las soluciones de Microsoft Active Directory, Novell Directory Services, ahora rebautizado a Novell e-Dyrectory, Red Hat Directory Server, Oracle Internet Directory o IBM Directory, por citar las soluciones más implantadas, han hecho que los árboles LDAP sean pieza fundamental en la seguridad del sistema y objetivo en muchos de los ataques.

Mirando todas las vulnerabilidades descubiertas en OpenLDAP a lo largo de su historia sólo se contabilizan, a día de hoy, 15 expedientes de seguridad, según la base de datos de Security Focus. Esto se puede deber a la alta calidad de los programas que lo han desarrollado o a un escrutinio poco estricto de la comunidad investigadora.

Este comportamiento se repite a lo largo de todas las implementaciones comerciales, luego se puede concluir que, hasta el momento los árboles LDAP no han estado sometidos a un estricto escrutinio por parte de la comunidad investigadora.

A continuación, se citan una lista de los ataques conocidos hasta la actualidad y su solución a nivel de seguridad.

Ataque 1. Ataque de Replay. Autenticación doble en entornos Pre-Shared Key.

Para este ataque hemos de suponer un entorno en el que los clientes se estén autenticando contra el árbol LDAP utilizando SASL S/Key, es decir, todos los clientes conocen una clave compartida. El atacante busca utilizar a un cliente como generador de respuestas ante desafíos emitidos por el servidor para conseguir una conexión autenticada sin tener la clave compartida.

En este entorno un atacante consigue acceso mediante el uso de la Pre-Shared Key de un usuario legítimo del sistema. Esta explotación se realiza aprovechándose de herramientas para realizar ataques Man In The Middle en redes inseguras como Ettercap o Yersinia, por poner algún ejemplo.

Esta técnica puede combinarse con el Ataque 2, que realiza un downgrade del sistema de autenticación negociado entre el cliente y el servidor, para conseguir que el método a utilizar sea el de Pre-Shared Key.

El siguiente diagrama muestra el proceso esquemático de envió de mensajes en un proceso de autenticación doble.

Figura 2.2-4: Ataque 1

El programa detectaría un fallo en la comunicación ya que detectaría que el servidor está respondiendo a la "Petición Conexión 2" del atacante MITM (Man In The Middle) y no a la "Petición Conexión 1" del Cliente utilizando el campo messageID por ejemplo.

El uso de entornos LDAP-s, dónde los participantes de la comunicación van autenticados con certificados digitales y las conexiones son cifradas con SSL, en dónde se produzca una correcta comprobación de certificados y el uso de comunicaciones basadas en el protocolo IPSec, siempre y cuando el protocolo IPSec no use Pre-Shared Key, evita la posibilidad de este tipo de ataques.

Ataque 2: Downgrading de Autenticación

Uno de los mecanismos soportados por el protocolo GSSAPI es el intercambio de credenciales en texto claro (Texto plano). Este sistema de autenticación solo debe utilizarse en entornos muy seguros o como única medida de compatibilidad entre el Cliente y el Servidor.

De acuerdo con el mecanismo de negociación de descrito por el protocolo SASL, cuando se va a producir una autenticación, con anterioridad tiene lugar una negociación entre el Cliente y el Servidor para que estos elijan el método SASL a utilizar. Si utilizan GSSAPI, entonces se intercambian los mecanismos que soportan, entre los que puede estar el sistema de Texto plano.

Un atacante en medio puede realizar un ataque de downgrading y, aunque el Cliente y el Servidor soporten mecanismos más robustos, hacer que se configure como mecanismo de autenticación el envío de credenciales en texto plano. La forma de hacerlo es enviar al Cliente, como mecanismos de

autenticación GSSAPI que son soportados por el Servidor, únicamente el de texto plano. Esto haría que el cliente sólo tuviera como opción el mecanismo texto plano.

Utilizando un programa que permita configurar la conexión cliente LDAP para acceder al árbol y poder realizar consultas como en la siguiente imagen, se podría desarrollar el ataque.

Figura 2.2-5: Configuración Ataque 2

Una vez realizada la configuración del Cliente y proporcionado las credenciales se procedería a la conexión con el servidor LDAP y, tal y como se refleja en la figura siguiente, se obtendría como resultado el acceso al servicio de directorio, pudiendo acceder a todos los objetos para los que el usuario en cuestión tenga permiso de acceso.

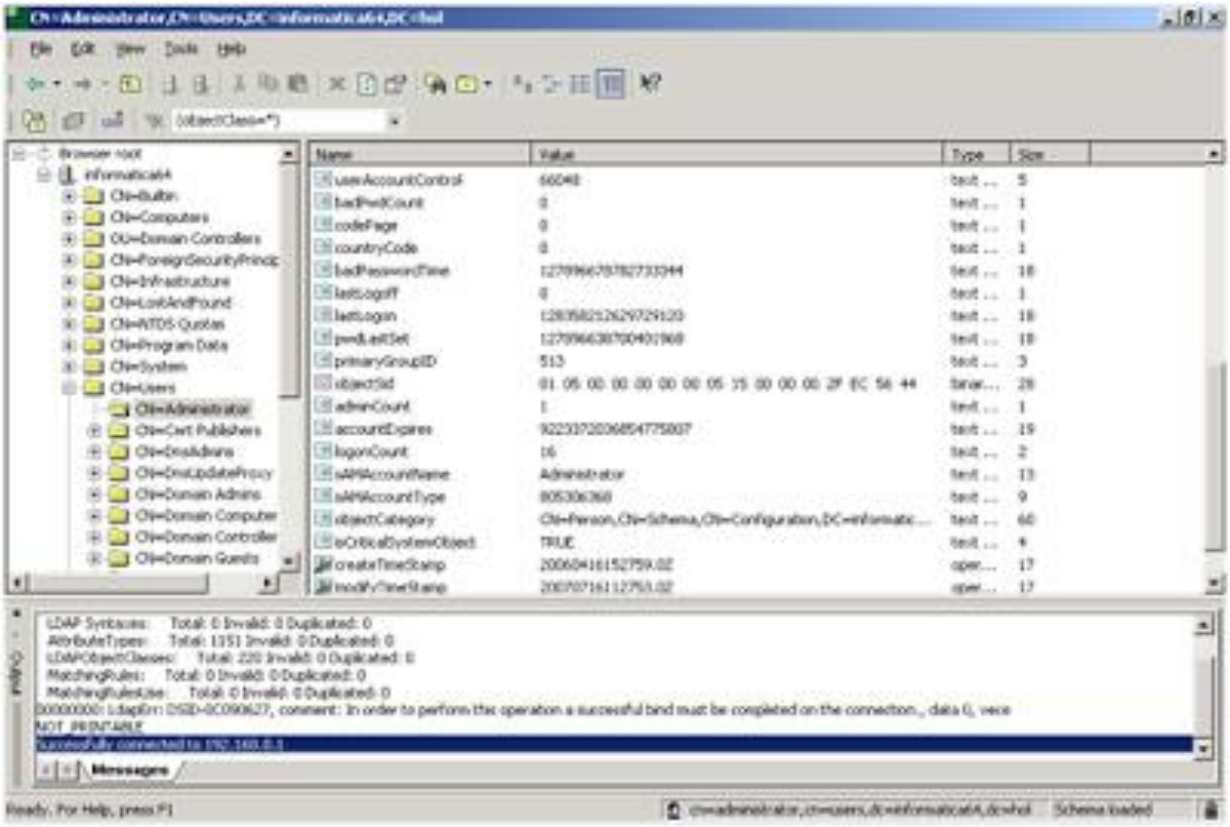

Figura 2.2-6: Acceso a directorio en Ataque 2

Las contramedidas, al igual que en el Ataque 1, consisten en evitar en este entorno la principal vulnerabilidad y lo que posibilita el ataque de downgrading, es decir, que se pueda producir un ataque de hombre en medio. Para evitarlo, la implantación de un sistema LDAP-s con comprobación segura de certificados digitales y el uso de sistemas IPSec, sin Pre-Shared Key, evitaría que se produjera este ataque.

Ataque 3: Captura de información transmitida

Una vez realizada la conexión entre el cliente y el servidor, se produce la transmisión de los datos solicitados por el cliente. Esta transmisión, por defecto se realiza sin utilizar ningún tipo de cifrado. Cualquier usuario que tenga acceso a las comunicaciones, mediante una conexión de red insegura podrá acceder a los datos.
Una red insegura puede ser cualquiera de los siguientes entornos:

- Redes Wifi: Las redes inalámbricas tienen la peculiaridad de estar al alcance cualquiera que esté cerca. Si la red inalámbrica no tiene autenticación o cifrado, o si el cifrado y autenticación que usa es WEP o si se está usando WPA-PSK y la contraseña no es robusta, es decir, de longitud corta, poca complejidad y no es cambiada cada cierto tiempo, entonces esa red Wireless es potencialmente insegura.

- Redes Cableadas: Si la red utiliza un concentrador de conexiones (hub) o usa switches sin IPSec o IPSec con Pre-Shared Key, sin detectores de Sniffers y sin detectores de técnicas envenenamiento (ARP-Poisoning) entonces esa red es potencialmente insegura.

En esas redes, el uso de Sniffers, o analizadores de tráfico, puede permitir a un atacante acceder a la información transmitida. La siguiente captura, realizada con el popular sniffer de red Wireshark en una red cableada con un concentrador de conexiones, ofrece al atacante los datos transmitidos tras la ejecución de una consulta de la carpeta USERS para ver su contenido.

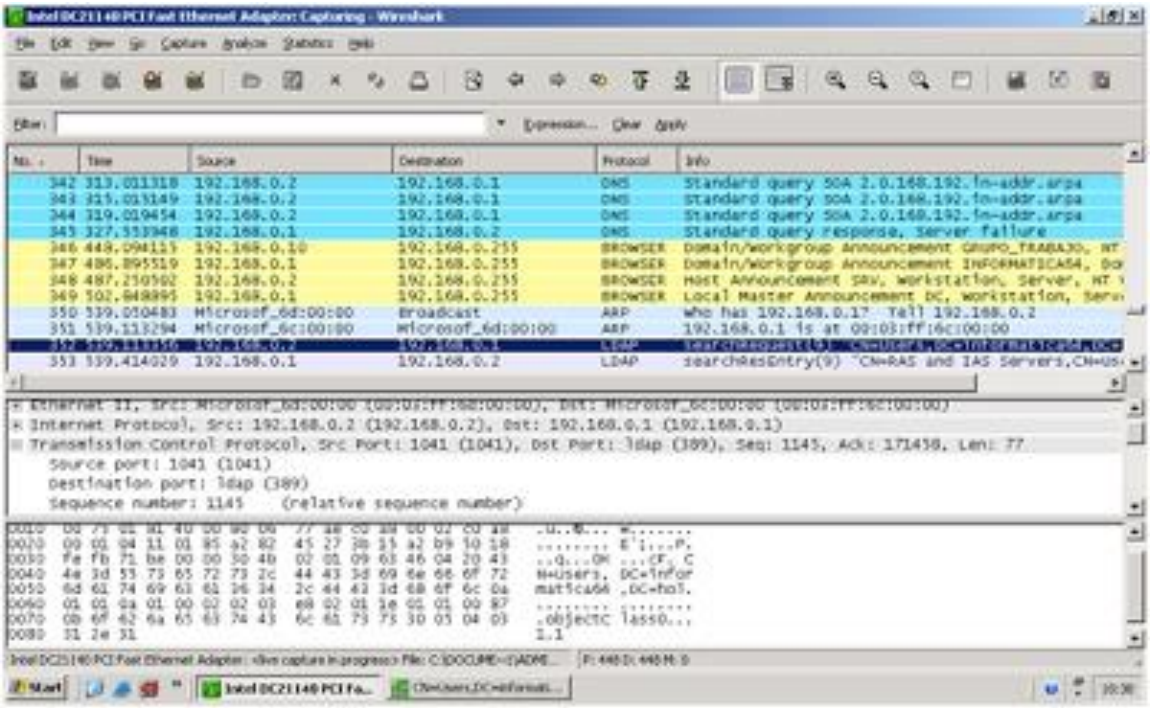

Figura 2.2-7: Ataque 3

Para evitar este tipo de ataques, reconocidos también en la RFC 4422, como ataques pasivos, se debe fortificar y cifrar el protocolo LDAP mediante el uso de LDAP-s y certificados digitales. No obstante, es posible realizar un ataque de Hijacking (ataque a la red donde el atacante toma control de la comunicación) de sesión, como se verá en el Ataque 4, mediante el uso de certificados digitales falsos si no existe una férrea comprobación de los certificados.

Ataque 4: Hijacking LDAP-s

La utilización de un sistema LDAP-s, es decir, encapsular las conexiones LDAP mediante un túnel SSL, puede ofrecer a los administradores la falsa tranquilidad y confianza en tener un sistema seguro. No obstante, el uso de esta tecnología no es garantía de éxito para conseguir evitar los ataques y existen pruebas de concepto públicas para atacar a estos entornos.

El uso de LDAP-s evita el Ataque 3 de acceso a los datos, sin embargo, es necesario, para evitar los Ataques 1 y Ataques 2, utilizar una comprobación correcta de los certificados digitales por parte del cliente, porque si no, se puede realizar un ataque de Hijacking.

En caso de no verificar correctamente los certificados digitales, la herramienta Cain&Abel, a partir de la versión 4.9.6, realiza Hijacking LDAP-s mediante técnicas de hombre en medio, es decir, cuando un Servidor LDAP-s envía el Certificado del Servidor al Cliente el atacante realiza una copia falsa del certificado, un duplicado.

Si el Cliente acepta certificados digitales no validados correctamente, entonces el atacante puede acceder a todos los datos de la conexión.

Esta comprobación depende de la configuración de seguridad que tenga el cliente o el uso que haga el usuario de la aplicación.

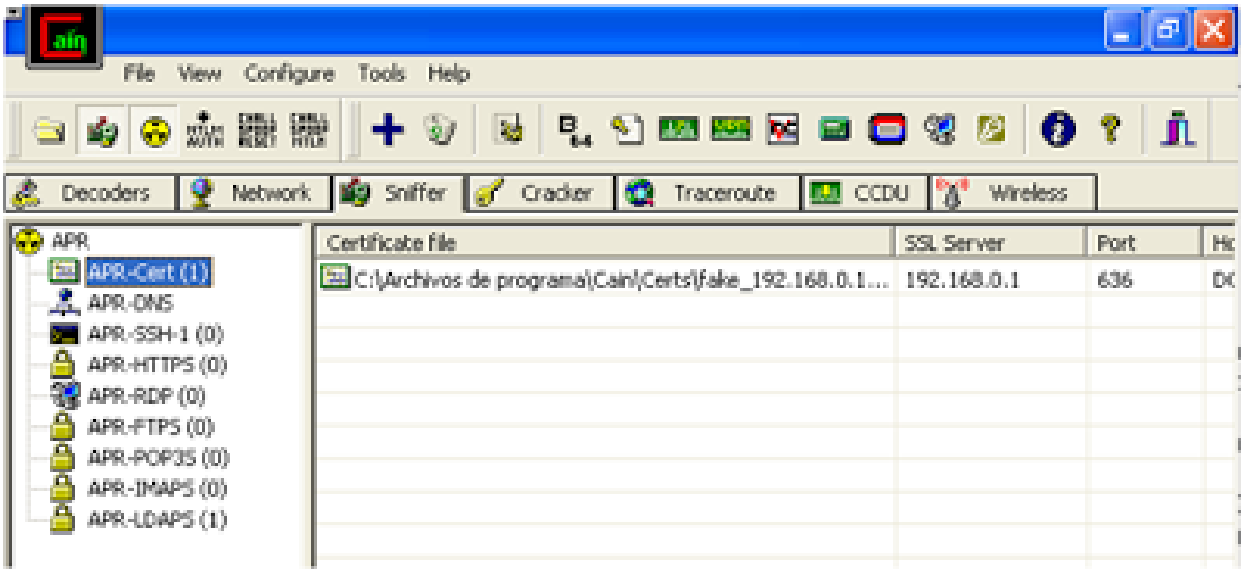

Figura 2.2-8 Creación de un certificado digital falso en el atacante usando Cain&Abel

A partir de ese momento, los clientes son los que deben fortalecer o no la seguridad del sistema. Clientes como LDP no permite la conexión en el momento que detecta un fallo en la validación del certificado.

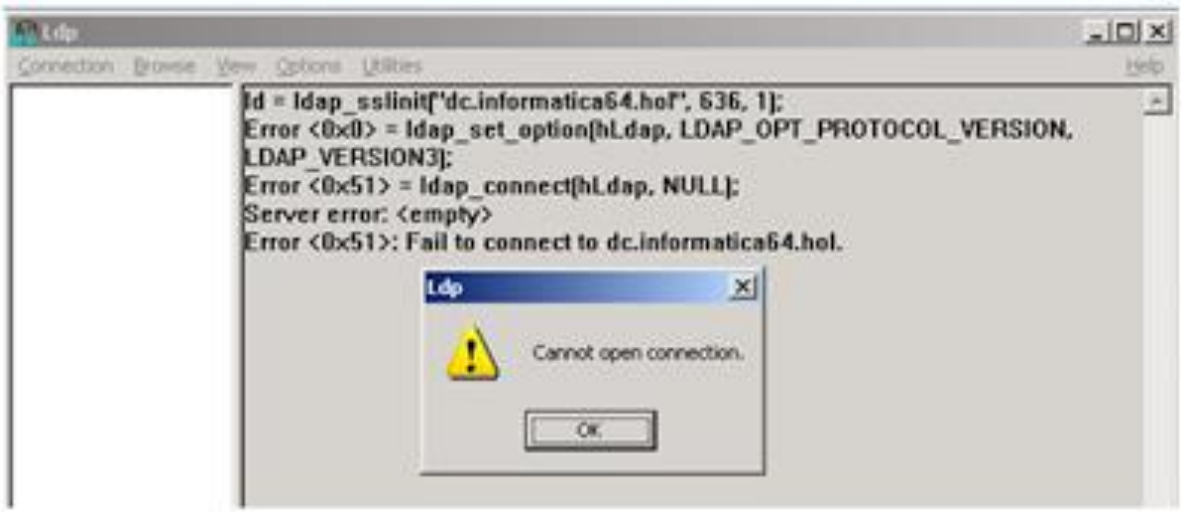

Figura 2.2-9 Conexión fallida con cliente LDP usando una conexión LDAP-s

Otros clientes, permiten la interacción del usuario generando una alerta, como se puede ver en la conexión realizada con el cliente LDAP de la fundación Mozilla[25]. Para ver el funcionamiento primero se debe configurar la conexión al servidor LDAP-s

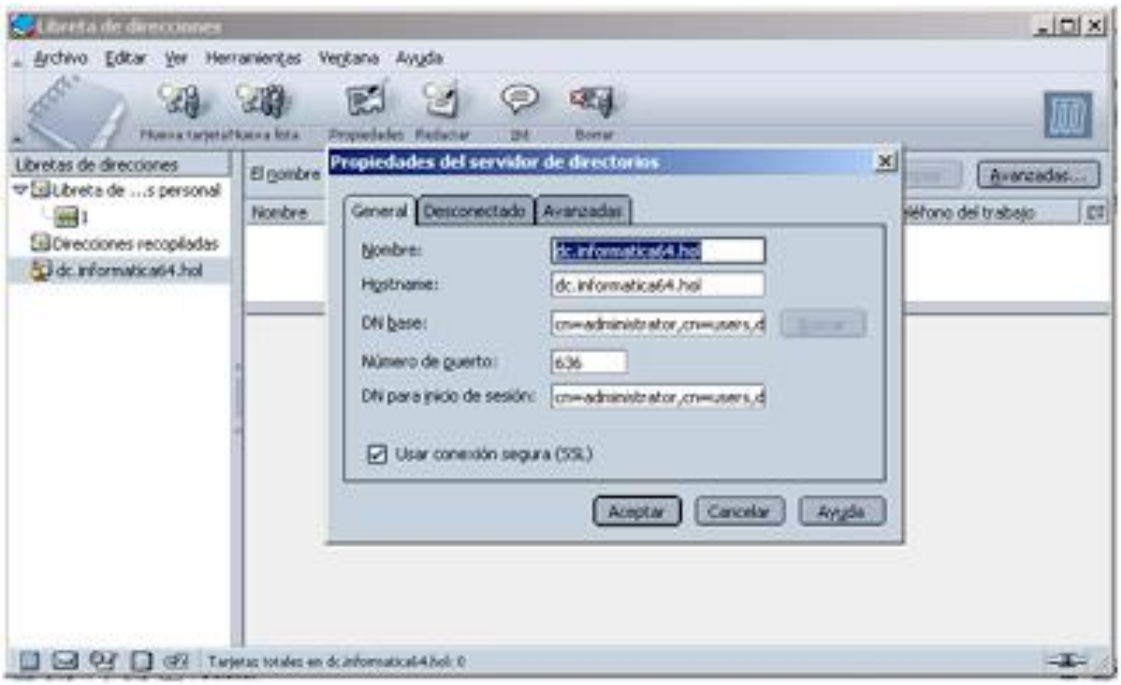

Figura 2.2-10 Configuración de la conexión LDAP-s

Una vez configurada se procede a realizar la conexión. El certificado que va a recibir el Cliente va a ser el duplicado falso generado por el atacante y el usuario recibirá una alerta de seguridad como se puede ver en la siguiente captura y tendrá que tomar una decisión de proseguir o no con la conexión LDAP-s utilizando dicho certificado.

Figura 2.2-11 Recepción de certificado falso y alerta

A partir de este punto es responsabilidad del usuario el aceptar o no el certificado digital. La no aceptación incurriría en una situación en la que no habría conexión, es decir, el atacante realizaría un ataque de Denegación de Servicio con éxito. En este entorno, esta sería la mejor opción, pues ningún dato quedaría en riesgo, incluyendo las credenciales del usuario.

En caso de aceptar la conexión, el cliente habría aceptado cifrar los datos con el certificado falso emitido por el hombre en medio, lo que le permitiría al atacante ver todos los datos, es decir, realizar el Ataque 3 y además, realizar un ataque de downgrading (Ataque 2), para poder acceder a las credenciales, lo cual implicaría el mayor riesgo de seguridad.

En la siguiente captura se puede ver como la herramienta de ataque CAIN&ABEL guarda en un fichero de texto todos datos transmitidos entre el Cliente y el Servidor. El Ataque 3 tiene éxito por tanto.

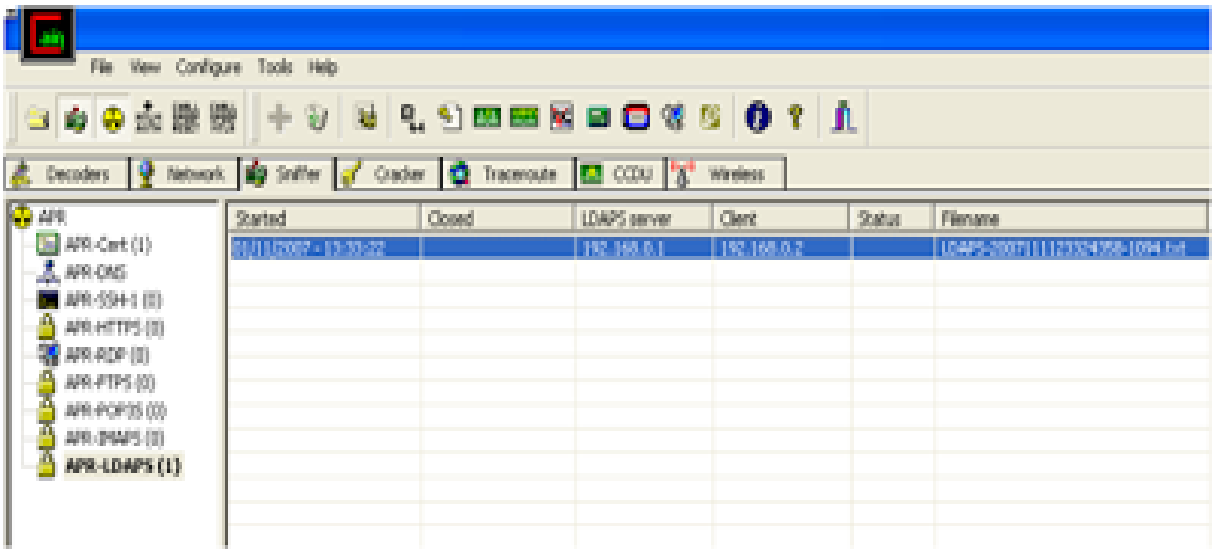

Figura 2.2-12 Hijacking y captura de sesión

En esta otra captura se puede observar como la contraseña es obtenida por el atacante tras ser aceptado por el cliente el certificado digital falso igual que si no se estuviera utilizando un certificado digital.

El Ataque 3, de downgrading, tiene éxito.

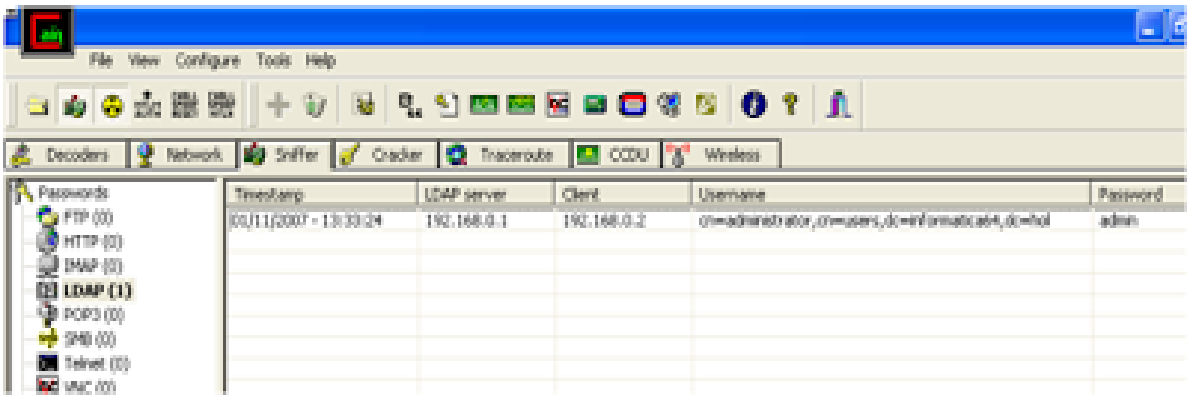

Figura 2.2-13 Credenciales obtenidas mediante un hijacking de sesión LDAP-s

Como se puede observar tras ver todos estos ataques, la única forma de mantener una infraestructura segura sería la de diseñar una red segura por ejemplo con conexiones IPSec sin Pre-Shared Key o mediante el uso de conexiones LDAP-s que no permitan la conexión utilizando certificados digitales falsos.

No obstante, detectar los fallos en la comunicación de todos los ejemplos que se han estudiado se conseguiría utilizando el programa desarrollado en el presente proyecto.

3 Diseño

El diseño del disector engloba las técnicas que se han implementado para conseguir un producto final cuya calidad cumpla con los requisitos de este proyecto. El producto no sólo ha de ser rápido, debe utilizar muy pocos recursos del sistema en el que se ejecute facilitando la adaptabilidad del cliente en cualquier tipo de entorno.

Para ello, a lo largo del proyecto, se han implementado una serie de técnicas de diseño que se enumeran a continuación y que, se explicarán en detalle:

- Disección TLV (Tipo Longitud Valor).
- Utilización de funciones de librería que no reservan espacio dinámico en memoria.
- Escritura de la salida del programa en disco mediante un archivo de texto.
- Puntero de caracteres a nivel de bit y máscaras.
- Scripts de comparación del programa con TShark mediante AWK.
- Listas enlazadas como solución para fragmentación de paquetes superior a 1460 bytes.
- Ejecución del programa en línea de comando mostrando barra de progreso y estadísticas.

3.1 Disección TLV

En el estudio del arte realizado en el apartado 2, se explica detalladamente la codificación del protocolo LDAP. Esta codificación es tipo BER (Basic Encoding Rules) siguiendo el estándar ASN.1 de la ITU-T.

Pues bien, el formato tipo BER especifica un formato de descripción y delimitación para las estructuras de datos de ASN.1. Cada dato está codificado como un identificador tipo, una descripción de la longitud, el valor del dato actual y, si fuera necesaria una marca de fin de contenido. Este tipo de codificación de dato se llama codificación TLV (Tipo Longitud Valor).

Identifier octets Type	Length octets Length	Content octets Value	End-of-contents octets

Figura 3.1-1 Representación TLV

El diseño del software mediante el uso de la codificación BER en formato TLV simplifica la operatividad y complejidad del sistema.

Esto ocurre porque se pueden utilizar las mismas funciones para cualquier tipo de dato ya que la codificación es genérica en todo el protocolo. Diseñando una función que capture el tipo adecuadamente, otra que calcule la longitud a partir del tipo capturado y, por último, una función que obtenga el valor del dato a diseccionar, se obtiene un disector que requiere una mecánica de disección sencilla.

3.2 Funciones de librería sin consumo de memoria

Como se ha comentado previamente, el disector se basa en funciones de código abierto. Muchas se han creado para este proyecto aunque las funciones que inician la conexión se han obtenido de las librerías de esta Universidad.

Es en las funciones que inician la conexión donde se ha notado una evolución del programa desde su inicio como un simple disector hasta el final donde se ha conseguido un programa óptimo.

Utilizando el comando "top", se comprobaba que el programa consumía memoria de manera muy rápida y creciente. Esto se debía a que las funciones "newEthFrameWithBytesTspec, newIPPacketWithEthFrame, newTCPSegmentWithBytes" que obtenían el tipo de fragmento "Ethernet, IP o TCP" reservaban memoria dinámica para alojar los datos. Estas funciones son útiles para disecciones de pocos paquetes que no requieran grandes iteraciones.

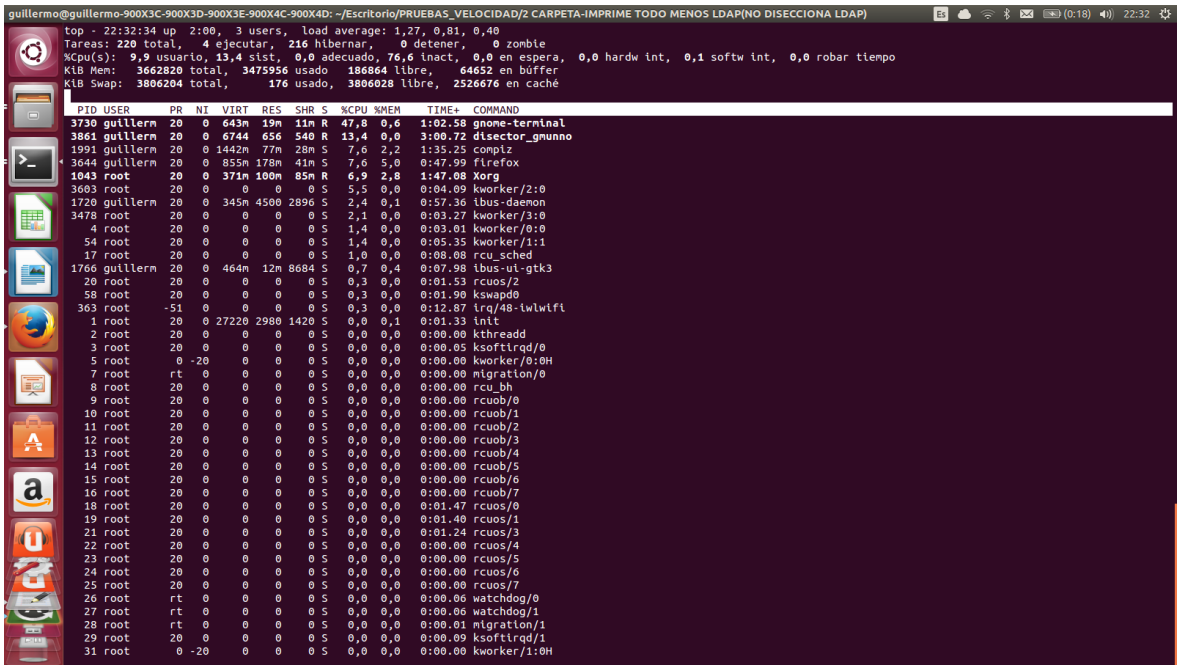

Figura 3.2-1: Monitorización del consumo de memoria con top

Se sustituyen por funciones con la misma funcionalidad de nombre similar pero con la adición del tag "_na" al final. La diferencia radica en que no se reserva memoria dinámica.

El problema de reservar memoria dinámicamente en el disector es que, procesaPkt crea un bucle de lectura de la trama analizando paquete a paquete la información y procesándola. Si la memoria dinámica no se libera posteriormente y, dado que se manejan ficheros de TBytes, esto supone un problema que impide la creación de un disector eficaz. Es más, no es necesario en este caso liberar memoria para inicializar las conexiones ya que todos los paquetes dentro de una trama poseen la misma estructura y con memoria estática se puede funcionar perfectamente.

3.3 Escritura en archivo

Durante el proyecto, se realizaban diversas pruebas de medición de consumo de memoria y procesado en ejecución como se ha mencionado previamente.

Inicialmente, estas pruebas que presentaremos más adelante medían el tiempo de ejecución del programa mostrando en un terminal de Linux la disección de la trama procesada. Utilizando tramas de 76 GBytes, se comprobaba que el tiempo de ejecución mostrando por pantalla el progreso podía llegar hasta los 3 días.

Esto se debe a que el proceso “gnome-terminal” consume por sí sólo memoria y procesado ya que, cuando se trata de ficheros muy grandes, almacena los datos para mostrarlos posteriormente.

Este proceso consumía memoria adicional al programa, lo cual dificultaba mucho un diseño óptimo. Sería poco eficiente disponer de un programa que consuma memoria y, que además requiera de un consumo adicional.

Para ello, se sustituye el mecanismo de representación en terminal por un mecanismo de escritura en archivo. Este mecanismo de escritura en archivo permite realizar mediciones en discos de lectura o discos de lectura/escritura.

La clave del diseño de una escritura que agilizara el proceso fue realizar un volcado previo de los datos que se iban a diseccionar posteriormente de manera que el sistema operativo primero leyera dichos datos en memoria y, posteriormente los procesara.

Como se observa en la figura siguiente, por un lado se almacena en disco los datos procesados y, por otro lado se ejecuta el programa permitiendo mayor velocidad de procesado.

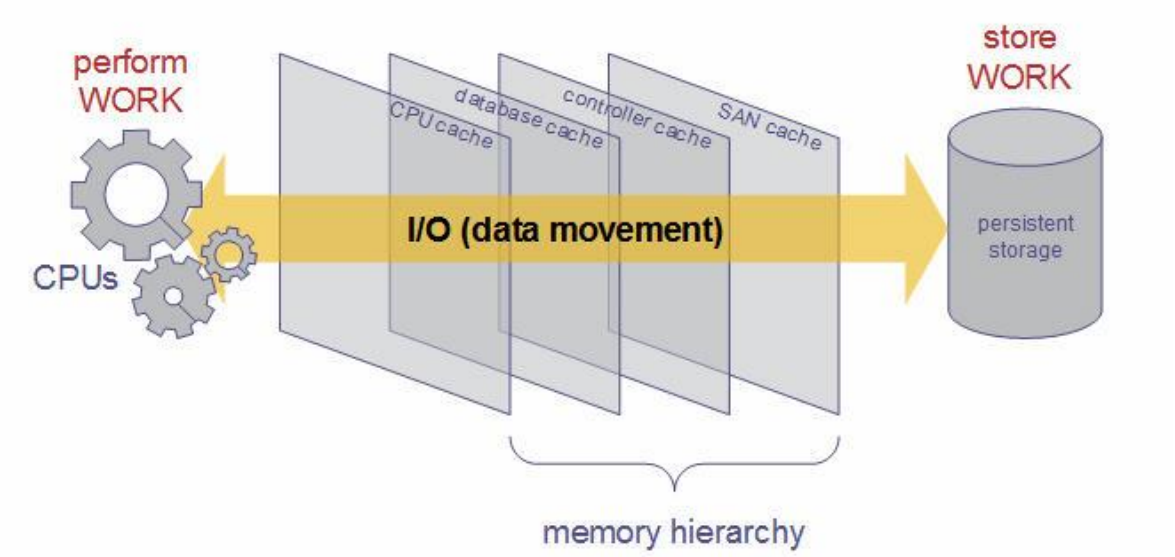

Figura 3.3-1: Almacenamiento y procesado

3.4 Punteros a nivel de bit y máscaras

Cada paquete almacena su información en un puntero que contiene la dirección de memoria relativa a los datos. Los datos que posee dicha dirección de memoria son bits que, en función del interés del usuario, pueden representarse de diversas formas con el fin de entender mejor su contenido.

El problema que se tuvo en un inicio fue la necesidad de representar estos bits en su forma hexadecimal con el motivo de poder comparar su salida con comparadores hexadecimales de byte.

Binary	Hex	Decimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Figura 3.4-1: Binario, Hexadecimal y Decimal

Para la representación de los bits de cada paquete se copiaba el contenido del puntero a una variable que almacenara dicho contenido pero en formato hexadecimal.

El proceso de copia suponía duplicar el tiempo de ejecución del programa ya que, se copiaba dos veces el mismo contenido simplemente para representarlo de manera más intuitiva.

La solución final fue trabajar con el contenido del puntero de los datos de cada paquete a nivel de bit y, compararlo utilizando máscaras de bits.

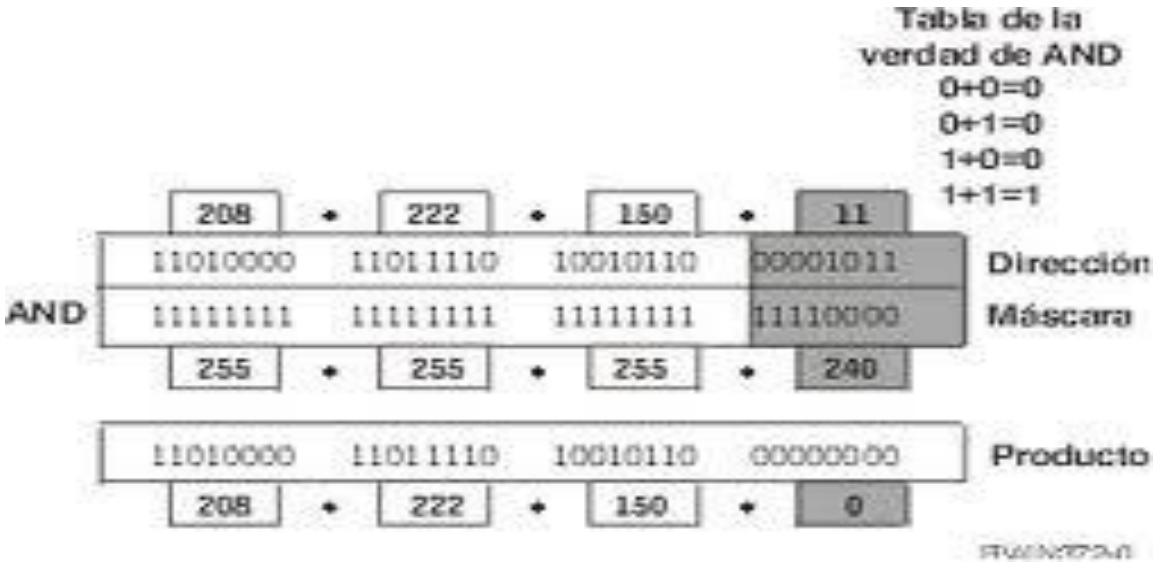

Figura 3.4-2: Máscaras y direcciones

Las máscaras fueron necesarias ya que para diseccionar adecuadamente se necesita partir el contenido de la posición del dato en el puntero. El dato del puntero, en cada posición, ocupa 1 byte pero, se quería obtener el valor cada 4 bits.

Esto suponía dividir en dos partes el byte. Se conseguía mediante máscaras como se ilustra a continuación.

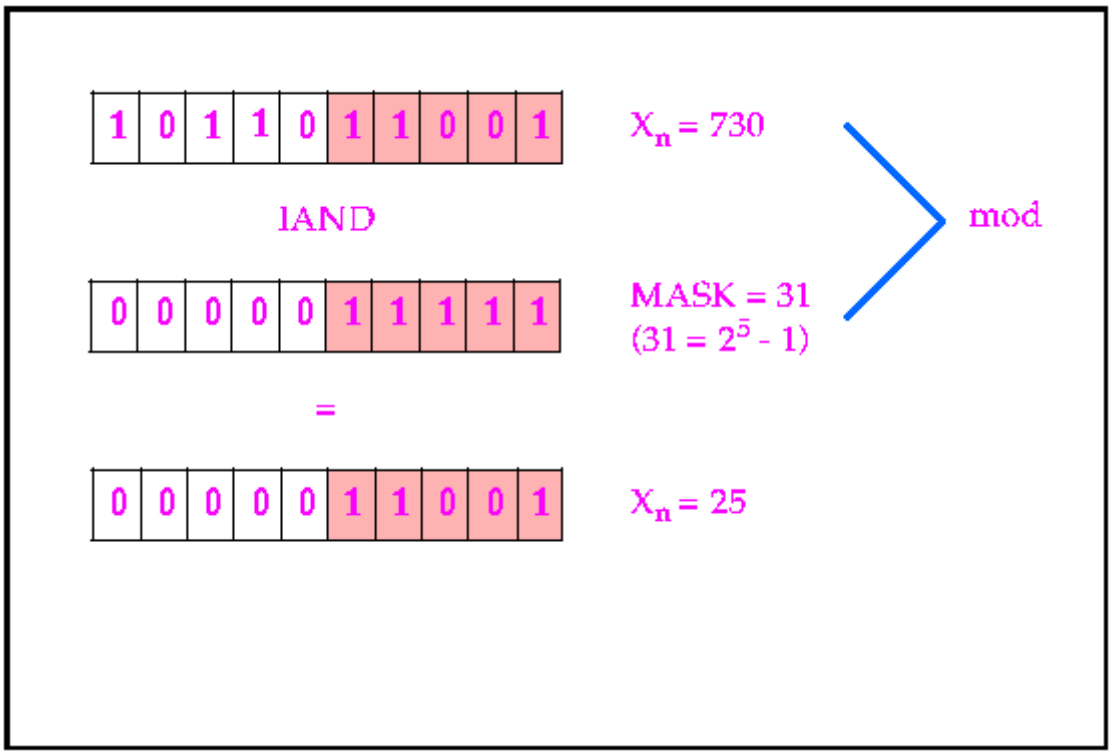

Figura 3.4-3: Máscaras para aislar 4 bits

3.5 Comparación salida con AWK

AWK es un lenguaje de programación diseñado para procesar datos basados en texto, ya sean ficheros o flujos de datos. Este tipo de lenguaje de programación se utiliza en este proyecto de cara a comparar la salida del programa ejecutado con un programa de "sniffing" ampliamente conocido como Wireshark.

Más concretamente, se compara la salida del programa con la versión de Wireshark a nivel de comando llamada TShark. En la sección de desarrollo del programa se mostrarán partes del código utilizado para realizar la comparación pero se debe destacar la facilidad para comparar la salida y, su gran funcionamiento en la comparación.

3.6 Listas enlazadas

Debido a que la longitud del paquete que se está analizando puede ser variable y de gran tamaño, se recurre a la implementación de listas enlazadas como mecanismo de unión de la información en un mismo paquete.

La longitud máxima que las funciones proporcionadas por la librería de código abierto de la Universidad Autónoma de Madrid son capaces de almacenar en un mismo paquete son 1460 bytes. A partir de ahí, el dato se divide en fragmentos de ese tamaño hasta que el paquete presente una longitud restante inferior a ese número.

Por ejemplo, un paquete de longitud total igual a 3509 Bytes se procesaría de la siguiente forma.

1460 Bytes	1460 Bytes	589 Bytes

Figura 3.6-1: Paquete sin enlazar

Para que la información del paquete no se perdiera en cada iteración y, con el fin de tener toda la información en un mismo paquete para facilitar la disección y aumentar el tiempo de procesado, se recurrió a las listas enlazadas.

Una lista enlazada es una estructura de datos que consiste en una secuencia de nodos en los que se guardan campos de datos arbitrarios y la referencia, enlace o puntero al nodo anterior o posterior.

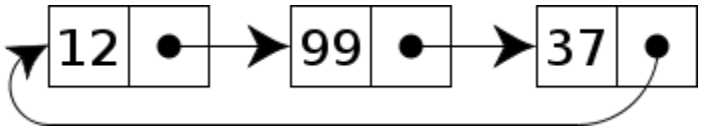

Figura 3.6-2: Listas Enlazadas

Ya que el sistema operativo que ejecuta el programa no sabe por sí sólo en qué posición de memoria se encuentra el dato una vez que dicha posición de memoria se ha dejado de acceder, se debe almacenar dicha dirección en una lista enlazada para no perder el datos que se necesite utilizar.

Debido a que se necesita reservar memoria dinámica ya que el número de bytes del paquete analizado es variable, es fundamental recorrer la lista una vez terminada la acción para liberar la memoria reservada previamente.

Con el uso de las listas enlazadas, se obtendría un paquete unificado de la siguiente manera:

3509 Bytes

Figura 3.6-3: Paquete con listas enlazadas

3.7 Ejecución del programa en línea de comando

El diseño del programa finaliza con la implementación de un mecanismo de ejecución amigable para el cliente. Se necesita únicamente una línea donde se especifican los comandos que se quieren obtener, si el cliente quiere mostrar por pantalla el progreso de la disección, si se quieren obtener estadísticas relativas a la ejecución y, en caso de dudas, se muestra una ayuda para los campos que se necesiten especificar.

Utilizando "optarg" como función dentro de la librería de funciones "getopt" se consigue diseñar un programa que obtenga los parámetros y los procese en función del tipo especificado.

"Getopt" es una librería de funciones en C que permite procesar opciones especificadas por línea de comandos.

"Optarg" es una función dentro de la librería "getopt" que devuelve un puntero con la cadena de caracteres del campo que ha conseguido procesar en la línea de comandos.

En la sección de desarrollo se mostrará un ejemplo del uso del programa como línea de comandos y, de los campos que "optarg" obtendría y procesaría para conseguir la ejecución final.

Finalmente, se muestra una barra de progreso de la disección que facilita al usuario la estimación del tiempo de lectura del archivo.

4 Desarrollo

El desarrollo del proyecto se centra en mostrar las técnicas que se han ido aplicando al programa y que se han detallado en el diseño del proyecto.

La primera etapa consiste en el desarrollo de un **disector simple. P**osteriormente se diseña una función que detecte el **mensaje tipo llamada "isLDAP()"**. Si el mensaje tipo es LDAP se pasa a diseccionar el paquete utilizando la **codificación TLV**.

En el desarrollo de la disección del paquete pueden aparecer paquetes de longitud superior 1460 bytes, por lo que se implementa un algoritmo de **listas enlazadas.**

Una vez desarrollada la disección, se perfilan las opciones de ejecución sobre **línea de comandos** para **comparar** finalmente su salida con **TShark mediante AWK** y verificar el resultado final del proyecto.

Para una mayor comprensión, se mostrarán las funciones y partes del código relevantes a su implementación desde el principio de la elaboración del proyecto hasta que se obtiene el resultado final.

4.1 DISECTOR SIMPLE

El punto de partida del desarrollo del proyecto fue la implementación de un disector sencillo que obtuviera los campos básicos y generales de cualquier protocolo.

Estos campos son:

- Tiempo de disección del paquete en formato Timestamp ().
- IP origen y destino.
- Puerto TCP origen y destino.
- MAC origen y destino.

Para ello, se utilizan las funciones de código abierto de la Universidad. El uso de las funciones utilizadas para obtener estos campos se describe a continuación:

```
1.  frame = newEthFrameWithBytesTspec_na((void*)bytes, header->len,
header->caplen, (struct timespec*)&header->ts, &frame_t);
2.  ipPkt = newIPPacketWithEthFrame_na(frame, &ipPkt_t);
3.  ipData = (void*)getIPData(ipPkt, &bufSize, &ipPktSize);
4.  sgmt = newTCPSegmentWithBytes_na(ipData, ipPktSize, bufSize,
&sgmt_t);
```

Figura 4.1-1: Funciones inicialización sin consumir memoria

Cabe destacar que estas funciones no consumen memoria y cumplen con el criterio de diseño definido previamente.
Con la función 1 se obtiene la cabecera Ethernet del paquete procesado, posteriormente se procesa su cabecera IP con las funciones 2 y 3.
Una vez obtenido el dato IP, se decodifica la parte correspondiente a la cabecera TCP obteniendo el segmento necesario para el puerto origen y destino.

A partir de los datos procesados, se utilizan las siguientes funciones para extraer los campos del disector simple final.

```
5.  cnxID->srcIP = getIPSrc(ipPkt);
6.  cnxID->dstIP = getIPDst(ipPkt);
7.  cnxID->srcPort = getTCPSrcPort(sgmt);
8.  cnxID->dstPort = getTCPDstPort(sgmt);
9.  cnxID->srcMAC=getClientSrcMac(frame);
10. cnxID->dstMAC=getClientDstMac(frame);
11. cnxID->tstamp=getIPPacketTimestamp(ipPkt);
```

Figura 4.1-2: Obtención de los campos del disector simple

De esta manera, se posee una estructura sencilla y organizada con los campos que se necesitaban obtener.

4.2 FUNCIÓN ISLDAP

La función que detecta si el paquete que se está procesando es LDAP es fundamental. Esto se debe a que si la función no detectara adecuadamente el tipo de mensaje que se transfiriese, se perdería un gran tiempo en diseccionar paquetes que no deberían haber sido diseccionados. Por tanto, permite ahorrar mucho tiempo de ejecución al sistema.

Se manejan tramas del orden de TBytes y, sí y sólo sí el paquete es LDAP, se procesan los campos del paquete en cuestión.

A continuación, se muestra una parte del código de la función que permite distinguir un paquete LDAP de otro cualquiera y se comenta brevemente su funcionalidad.

```
1.      int isLDAP( char *dtosTCP){
2.
3.              if(dtosTCP[0]==0x30){
4.
5.                      if((dtosTCP[1]&0xf0)==0x80){
6.
7.                      long_tlb=dtosTCP[1]&0x0f;
8.                      long_messageid=dtosTCP[3+(long_tlb)]&0x0f;
9.
10.                     switch((dtosTCP[4+(long_tlb)+
11.                     (long_messageid)]&0x0f)){
12.
13.                     case 0:   return 1;break;
14.                     case 3:  return 1; break;
15.                     case 7:  return 1; break;
16.                     case 8:   return 1; break;
17.                     default: return 0;break;
18.
19.                     }
20.
21.                     }
22.
23.                     return 1;
24.             }
25.             else if((dtosTCP[4+(long_tlb)+
26.             (long_messageid)]&0xf0)==0x40){
27.
28.                     return 1;
29.             }
30.             return 0;
31.     }
```

Figura 4.2-1: Función isLDAP()

La función básicamente devuelve 1 si, tras analizar los campos utilizando máscaras a nivel de bit, detecta que el mensaje es tipo LDAP. En caso contrario, devuelve 0.

4.3 DISECCIÓN LDAP TLV

Ya se ha explicado en el diseño en qué consiste este tipo de codificación. A continuación, se muestran la función que hace posible obtener los campos relevantes al protocolo:

```
1.	int eslongitud(){
2.
3.		if((dtosTCP[posicionmac+1]&0xf0)!=0x80){
4.
5.		longitud=hexadecimal(dtosTCP[posicionmac+1]&0xf0,1)
6.		+hexadecimal(dtosTCP[posicionmac+1]&0x0f,0);
7.
8.		*nuevaposicion=posicion+6;
9.		*nuevaposicionmac=posicionmac+2;
10.
11.		}
12.
13.		else{
14.
15.		j=dtosTCP[posicionmac+1]&0x0f;
16.		x=j;
17.
18.		for(i=0;i<j;i++){
19.
20.		longitud=hexadecimal(dtosTCP[posicionmac+2+(i)]&0xf0,
21.		 (1+(2*(x-1))))+
22.		hexadecimal(dtosTCP[posicionmac+2+(i)]&0x0f,
23.		(2*(x-1)))+longitud;
24.
25.		x--;
26.
27.		}
28.
29.		*nuevaposicion=posicion+6+(3*j);
30.		*nuevaposicionmac=posicionmac+2+j;
31.
```

```
32.                 }
33.
34.       return longitud;
35.
36.       }
```

Figura 4.3-1: Función eslongitud()

Esta función permite obtener la longitud del dato previa obtención del tipo con una comparación a nivel de máscaras.

```
1.        void esvalor(){
2.
3.         if((parametro2==2)||(parametro10==10)){
4.
5.        for(i=0;i<(longitud);i++){
6.
          formatinteger=hexadecimalint(mac[nuevaposicionmac+i]
7.        &0xf0,1+2*(x-1))+
8.        hexadecimalint(mac[nuevaposicionmac+i]
9.        &0x0f,
10.       2*(x-1))+formatinteger;
11.
12.       x--;
13.
14.       }
15.
16.       fprintf(stream,"%d",formatinteger);
17.
18.       }
19.
```

Figura 4.3-2: Función esvalor()

Con la función esvalor() se procesa el valor del tercer argumento TLV obteniendo el contenido del campo a diseccionar.

A continuación, se muestra un ejemplo de la disección de un campo integrando una comparación para obtener el tipo de dato mediante el uso de máscaras de bits y las funciones para calcular su longitud y obtener el valor final.

```
1.      longitud=eslongitud();
2.
3.      if((parametro==1)||(parametro7==7)){
4.
5.      if(verbose==1)fprintf(stream,"   AUTHENTICATION|   ");
6.
7.
        if(((dtosTCP[posicionfinalmac]&0x0f)==0x00)&&(parametro==1))
8.
9.      fprintf(stream," SIMPLE          ");
10.
11.     else  if(((dtosTCP[posicionfinalmac]&0x0f)==0x03)
12.     &&(parametro==1)){
13.
14.     fprintf(stream," SASL   ");
15.
16.     esvalor();
17.
18.     }
19.
20.             posicionfinal=nuevaposicion+(3*longitud);
21.             posicionfinalmac=nuevaposicionmac+longitud;
22.     }
```

Figura 4.3-3: Ejemplo disección campo Authentication

El ejemplo muestra el campo “Authentication” y, en función de si es un tipo u otro, disecciona la salida del protocolo.
Primero se obtiene la longitud inicial con “eslongitud()” para situar la posición en el puntero del paquete LDAP, posteriormente se compara la posición de este puntero con una máscara a nivel de bit representada en hexadecimal.

Una vez obtenido el tipo de campo, se procesa el valor del campo utilizando la función “esvalor()”.
Finalmente se posiciona el puntero en la siguiente posición para seguir diseccionando de forma adecuada.

4.4 LISTAS ENLAZADAS

Una vez explicados los fundamentos de las listas enlazadas y el motivo de su uso en el apartado de diseño, se muestra a continuación la definición de la estructura y su uso en el disector.

```
1.      typedef struct _nodo{
2.              char fragmentLDAP[1460];
3.              struct _nodo *siguiente;
4.      }nodo;
5.
6.      nodo *primero;
7.      nodo *ultimo;
8.      nodo *nuevo;
9.
10.     primero=(nodo *)NULL;
11.     ultimo=(nodo *)NULL;
12.     nuevo=(nodo *)NULL;
```

Figura 4.4-1: Estructura de listas enlazadas

Se define una lista enlazada con el valor del paquete LDAP con un tamaño máximo de 1460 bytes y un puntero a la siguiente estructura. Después se declaran las variables a utilizar en la disección.

A continuación se muestra el desarrollo de las listas enlazadas en el disector.

```
1.              nuevo=(nodo *)malloc(sizeof(nodo));
2.              memcpy(nuevo->fragmentIP,datosTCP,newSize);
3.              nuevo->siguiente=NULL;
4.
5.              if(primero==NULL){
6.
7.                              primero=nuevo;
8.                              ultimo=nuevo;
9.              }
10.             else{
11.                             ultimo->siguiente=nuevo;
12.                             ultimo=nuevo;
13.             }
14.
15.             if(isLDAP){
```

```
16.
17.                                     //DISECCIÓN ...
18.
19.                             liberar(primero);
20.                             liberar(ultimo);
21.                             free(nuevo);
22.                             primero=NULL;//ultimo=NULL;
23.                             longfinal=0;
24.                             fprintf(stream, "\n");
25.
26.                     }
27.                     else{
28.                             primero=NULL;
29.                             free(nuevo);
30.                     }
31.
32.     }
```

Figura 4.4-2: Función implementa listas enlazadas

Se obtiene la cabecera LDAP con la función “getTCPData()”, para posteriormente crear la lista enlazada con el contenido del paquete LDAP que se procesa en cada iteración. Una vez diseccionado el paquete, se recorre la lista para liberarla y pasar a procesar el siguiente paquete. La liberación de la lista la realiza la siguiente función:

```
1.      void liberar(nodo *primero){
2.
3.              nodo *aux=primero;
4.              while(aux->siguiente!=NULL)
5.              {
6.                      nodo *borrar;
7.                      borrar=aux;
8.                      aux=aux->siguiente;
9.                      free(borrar);
10.
11.             }
12.     }
```

Figura 4.4-3: Función libera memoria de listas enlazadas

La función recorre la lista enlazada hasta que el puntero de la lista tenga contenido y borra el espacio de memoria reservado previamente.

4.5 LÍNEA DE COMANDOS: OPTARG Y BARRA DE PROGRESO

En este apartado se va a concretar el contenido del desarrollo para diseñar un programa que ejecute el proceso de disección en una línea de comandos de la forma más sencilla e intuitiva para el usuario.

Se explicará cómo se obtienen los parámetros y, su posterior interpretación para procesarlos. También se detallará el desarrollo de la función que muestra la barra de progreso así como la opción que permite al usuario imprimir la salida en un archivo o mostrar estadísticas relativas al sistema.

Para que se comprenda el funcionamiento del programa se va a mostrar la ejecución del mismo en un terminal.

Figura 4.5-1: Disector en línea de comandos

Para leer los parámetros se utiliza la función "optarg" de la librería "getopt" que ya hemos descrito en el diseño del programa. Por tanto, se muestra a continuación su funcionamiento y desarrollo.

```
1.  while ((letra=getopt(argc, (char * const*)argv, "f:q:p:o:t:vhu0e"))!=EOF)
2.          {
3.                  switch (letra)
4.                  {
5.                          case 'h':
6.                                  ayuda(argv[0]);
7.                                  exit(0);
8.                          case 'e':
9.                                  estadisticas=1;
10.                                 break;
11.                         case 'p':
12.                                 fileFormat = optarg;
```

```
13.                             break;
14.                     case 'f':
15.                             pcapFilePath = optarg;
16.                             break;
17.                     case 'o':
18.                             outputFilePath = optarg;
19.                             break;
20.                     case 'q':
21.                             str=optarg;
22.                             if(str!=NULL){
23.
24.                             if(strstr(str,"MESSAGEID")!=NULL){
25.                             parametro2=2;
26.                             }
27.
28.                             if(strstr(str,"PROTOCOLOP")!=NULL){
29.                             parametro3=3;
30.                             }
31.
32.                             if(strstr(str,"RESULTCODE")!=NULL){
33.                             parametro10=10;
34.                             }
35.
36.                             }
37.                             break;
38.             }
39.}
```

Figura 4.5-2: Optarg como captura de parámetros

Como se acaba de comprobar, utilizando el comando optarg y, previa obtención de los comandos con getopt, se detecta la letra del parámetro deseado y se procesa la solicitud en función del tipo.

Cabe destacar que con el parámetro –o se escoge la opción de escritura en disco que recordamos ayudaba en la reducción de tiempo de procesado.

Se especifica el formato de las tramas que se analizan aunque por defecto se manejan formatos tipo “raw”, “pcap” y “cap”.
Con la opción -e se muestran las estadísticas como el número de paquetes procesados, el tipo de fragmento o su tamaño.

Si en la línea de comandos se especificara la opción -h, se mostraría una ayuda con las opciones y su contenido.

Por último, una ejecución del disector mostraría una barra de progreso similar a la siguiente:

```
1.  ./ldap_disector -f ../c06-ldapv3-app-r1.pcap -o output.txt
2.
3.                          /*      BARRA DE PROGRESO       */
4.
5.                          [==========================]
6.
7.                          /*      BARRA DE PROGRESO       */
```

Figura 4.5-3: Barra de progreso de lectura de fichero

4.6 AWK: Comparación con TShark

La última parte del desarrollo del proyecto es la verificación de su salida mediante su comparación con la salida que devuelve Tshark. Para ello, se utiliza el lenguaje de programación AWK basado en shell-scripting que se ha detallado previamente. En concreto se utiliza "gawk" como alternativa a AWK.

La idea de la comparación es sencilla, se restan las dos salidas y se devuelve el número del fragmento que no es igual. En el proyecto se desarrollan cinco scripts de comparación y, se realizan múltiples pruebas con tramas de gran tamaño y de distinto origen.

Los scripts de comparación se centran en mostrar las diferencias de los siguientes campos del protocolo:

1. **Número de mensajes**: el número de mensajes en ambas disecciones tienen que ser el mismo.

2. **Tipo de protocolo**: el tipo de protocolo debe ser similar. Cabe destacar que si se disponen de paquetes donde hay más de un protocolo, la disección del programa los muestra todos separados mediante comas de la misma manera que lo hace Tshark.

3. **Número de identificación de mensaje o messageID**: mediante la representación decimal de este campo, se comparan los números utilizando como siempre AWK.

4. **Código de resultado o resultCode**: de la misma manera que la comparación del campo messageID, se compara la salida en su formato decimal.
5. **Nombre canónico o objectname**: se compara la salida como cadena de caracteres.

Ya que la comparación de los campos 2, 3, 4 y 5 es similar, se va a mostrar sólo un ejemplo. La excepción es el número de mensajes que se compara simplemente comprobando que ambos números coinciden sin necesidad de AWK.

```
1. #!/bin/sh
2.
3. ./../ldap_disector -f $1 -q MESSAGEID -o output.txt
4.
5. tshark -r $1 -T fields -e frame.time_epoch -e ip.src -e ip.dst -e tcp.srcport -
   e tcp.dstport -e eth.src -e eth.dst -e ldap.messageID -e frame.number -E
   separator=- 'ldap' > tsharkoutput.txt
6.
7. gawk -F "-" 'FNR==NR{a[$2$3$4$5$6$7$8];next}!($2$3$4$5$6$7$8 in
   a){print $9}' output.txt tsharkoutput.txt
8.
```

Figura 4.6-1: Comparación de messageID

Obtención del número de mensajes mediante el comando “wc”:

```
1. ./ldap_disector -f ldap-and-search.pcap | wc -l
2.
3. tshark -r ldap-and-search.pcap 'ldap' | wc –l
4.
```

Figura 4.6-2: Comparación número de mensajes

Se puede dar por concluido el desarrollo del proyecto, a modo resumen se ha mostrado la evolución del programa reuniendo todos los aspectos argumentados en el diseño del proyecto.

5 Integración, pruebas y resultados

Los tres tipos de pruebas principales del proyecto han analizado el consumo de memoria, la velocidad de procesado y la efectividad.

5.1 Consumo de memoria

Se ha utilizado el programa Valgrind para medir dicho consumo con diversos ficheros. Este programa es de gran utilidad ya que proporciona información completa acerca de las líneas de código donde no se libera dicha memoria o el total de Bytes de memoria que se han perdido.

```
1.  Valgrind –leak-check=full ./ldap_disector -9 ../traza1.pcap
2.
3.  LEAK SUMMARY:
4.        Definitely lost: 1,120 bytes in 1 blocks
5.        Indirectly lost: 507 bytes in 2 blocks
6.        Possibly lost: 0 bytes in 0 blocks
7.        Still reachable: 66,239 bytes in 3 blocks
8.        Suppressed: 0 bytes in 0 blocks
9.
```

Figura 5.1-1: Valgrind primera trama

```
10. Valgrind –leak-check=full ./ldap_disector -9 ../traza2.pcap
11.
12. LEAK SUMMARY:
13.       Definitely lost: 1,120 bytes in 1 blocks
14.       Indirectly lost: 507 bytes in 2 blocks
15.       Possibly lost: 0 bytes in 0 blocks
16.       Still reachable: 65,887 bytes in 3 blocks
17.       Suppressed: 0 bytes in 0 blocks
```

Figura 5.1-2: Valgrind segunda trama

```
18. Valgrind –leak-check=full ./ldap_disector -9 ../traza3.pcap
19.
20. LEAK SUMMARY:
21.       Definitely lost: 1,120 bytes in 1 blocks
22.       Indirectly lost: 507 bytes in 2 blocks
23.       Possibly lost: 0 bytes in 0 blocks
24.       Still reachable: 67,154 bytes in 5 blocks
25.       Suppressed: 0 bytes in 0 blocks
```

Figura 5.1-3: Valgrind tercera trama

Como se puede comprobar con las tres pruebas realizadas se ve como la memoria perdida de manera definitiva, indirecta o posible es extremadamente pequeña lo que nos permite crear un programa sin fisuras y altamente eficiente. La memoria que aún se puede acceder es también muy pequeña ya que se analizan tramas enormes de millones de bytes.

5.2 Velocidad de procesado

Para medir la velocidad de procesado del programa se ha utilizado el comando time que calcula el tiempo que transcurre desde que se ejecuta el programa hasta que termina la disección. Se han realizado pruebas en discos de lectura y escritura.

```
Command being timed: "./disector_gmunnoz -f /home/automator/1391493600_1391580000_dup2.pcap"
    User time (seconds): 446.69
    System time (seconds): 278.74
    Percent of CPU this job got: 27%
    Elapsed (wall clock) time (h:mm:ss or m:ss): 44:22.15
    Average shared text size (kbytes): 0
    Average unshared data size (kbytes): 0
    Average stack size (kbytes): 0
    Average total size (kbytes): 0
    Maximum resident set size (kbytes): 3280
    Average resident set size (kbytes): 0
    Major (requiring I/O) page faults: 0
    Minor (reclaiming a frame) page faults: 251
    Voluntary context switches: 805319
    Involuntary context switches: 106000
    Swaps: 0
    File system inputs: 1076821712
    File system outputs: 0
    Socket messages sent: 0
    Socket messages received: 0
    Signals delivered: 0
    Page size (bytes): 4096
    Exit status: 0
```

Figura 5.2-1: Velocidad de procesado primera trama

Como se aprecia en esta prueba de velocidad, se analiza un fichero de tamaño 1.076.821.712 bytes (1,076 TBytes) en 44 minutos y 22 segundos lo que supone un procesado de 1.076 GBytes / (44*60 + 22) segundos = **0,4042 Gbps**.

```
1  time ./ldap_disector -9 ../s5.pcap -o mifichero.txt
2  [==========================]
3
4  real    12m11.827s
5  user    4m4.555s
6  sys     4m12.032s
```

Figura 5.2-2: Velocidad de procesado segunda trama

En esta prueba se mide la velocidad del programa con un fichero de 76 GBytes en disco de estado sólido o SSD. Se tardan 12 minutos en analizar 76 GBytes lo que supone una velocidad de procesado de 76 GBytes / (12*60 + 11) = **0,1039 GBps**

```
time ./ldap_disector -9 ../traza_prueba_sin_binds.pcap -o mifichero.txt
[==========================]

real    0m21.827s
user    0m4.555s
sys     0m12.032s

```

Figura 5.2-3: Velocidad de procesado tercera trama

La prueba se realiza sobre un fichero con disco duro de escritura limitado por la máquina virtual. En este caso tenemos un fichero de 54,6 MBytes y tarda 21 segundos en analizarlo lo que supone una velocidad de procesado de 0,0546 MBytes / 21=**0,0026 Gbps**.

Se puede concluir que la velocidad de procesado es mayor si se reserva un disco duro para sólo la lectura y otro para escritura en una gran máquina. Con una máquina inferior pero con disco de estado sólido o SSD el tiempo también es extremadamente bueno. Por último, si la máquina posee limitaciones la velocidad es inferior aunque es óptima.

5.3 Efectividad con Wireshark

Las pruebas de la efectividad del programa han recogido múltiples análisis. Se ha calculado la diferencia de los resultados que Wireshark mediante su variante TShark han sido capaces de almacenar y los resultados que el disector proporcionaba en los cinco puntos que se han comentado previamente. Estos son los porcentajes de efectividad.

Diferencia de número de mensajes:

Utilizando varias tramas se ha visto como el número de mensajes es altamente parecido mostrando un porcentaje de acierto como el siguiente:

Porcentaje de aciertos en el **número de mensajes** 845297/846560 = 0.9985 = **99,85 %**

Porcentaje de aciertos en el **número de mensajes** 11939/12688 = 0.9408 = **94,08 %**

Porcentaje de aciertos en el **número de mensajes** 9114/9205 = 0.9901 = **99.01 %**

Diferencia de número de campo messageID:

Porcentaje de aciertos en el **campo messageID** 845461/846560 = 99,87 %

Porcentaje de aciertos en el **campo messageID** 12376/12688 = 0.9754 = **97,54 %**

Porcentaje de aciertos en el **campo messageID** 9003/9205 = 0.9780 = **97,80 %**

Diferencia de número de campo protocolOp:

Porcentaje de aciertos en el **campo protocolOp** 845464/846560 = **99,87 %**

Porcentaje de aciertos en el **campo protocolOp** 12356/12688 = 0.9738 = **97,38 %**

Porcentaje de aciertos en el **campo protocolOp** 9003/9205 = 0.9780 = **97,80 %**

Diferencia de número de campo resultCode:

Porcentaje de aciertos en el **campo resultCode** 846351/846560 = **99,97 %**

Porcentaje de aciertos en el **campo resultCode** 12376/12688 = 0.9757 = **97,57 %**

Porcentaje de aciertos en el **campo resultCode** 9006/9205 = 0.9783 = **97,83 %**

Diferencia de número de campo objectName:

Porcentaje de aciertos en el **campo objectName** 684892/846560 = **80,90 %**

Porcentaje de aciertos en el **campo objectName** 12380/12688 = 0.9757 = **97,57 %**

Porcentaje de aciertos en el **campo objectName** 9003/9205 = 0.9787 = **97,87 %**

Como se puede comprobar hay un altísimo número de aciertos en todos los campos. Cabe destacar que el campo objectName no tiene el elevado número de aciertos que sí ofrecen los otros campos en el primer análisis debido a que Wireshark utiliza un reensamblado que se escapa del alcance de este proyecto de fin de carrera. Algunos paquetes se reensamblan siguiendo algoritmos que no podemos utilizar con las librerías de código abierto que hemos utilizado.

En el resto de los casos el disector es de una eficiencia extrema.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

El proyecto ha conseguido estudiar el comportamiento del protocolo LDAP en una comunicación genérica.

Para ello, se ha desarrollado un programa de altas prestaciones capaz de obtener unos resultados de efectividad, consumo y procesado que permiten al usuario que esté en disposición de utilizarlo una mayor facilidad de uso en el entorno en el que se encuentre.

Se han estudiado diversos escenarios y analizado cómo se utilizaría el programa para solucionar posibles fallos en la transmisión del protocolo.

Esto significa que no sólo se ha realizado un estudio del estado del arte del protocolo sino que se ha otorgado una posibilidad de solventar fallos en la comunicación lo que supondría un gran beneficio a nivel económico para una empresa.

6.2 Trabajo futuro

Se abren varias líneas de investigación para el futuro:

- **Reensamblado automático** de la cabecera TCP obteniendo un único paquete LDAP.
- Script de **detección automática** de **fallos** en la comunicación.
- **Solución** ante **ataques de personas ajenas** al programa que deseen obtener información de contactos dentro del directorio.
- Solución ante **Hijacking** o **Downgrading**.
- Disección de **múltiples tramas** en entornos **Windows** con **distintas arquitecturas.**

- **Integración** con la comunidad **openLDAP** ofreciendo un disector de código libre.

Referencias

[1] Brian Kernighan, and Dennis Ritchie, "The C Programming Language 2nd Edition", http://www.tecnica1lomas.com.ar/tutoriales/lenguaje_C.pdf

[2] K. Zeilenga, Ed. OpenLDAP Foundation, "Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map", https://tools.ietf.org/html/rfc4510

[3] J. Sermersheim, Ed. Novell Inc., "Lightweight Directory Access Protocol(LDAP): The Protocol", https://tools.ietf.org/html/rfc4511

[4] K.Zeilenga, Ed. OpenLDAP Foundation, "Lightweight Directory Access Protocol(LDAP): Directory Information Models", https://tools.ietf.org/html/rfc4512

[5] R. Harrion, Novell Inc, "Lightweight Directory Access Protocol(LDAP): Authentication Methods and Security Mechanisms", https://tools.ietf.org/html/rfc4513

[6] OpenLDAP, "Using SASL", http://www.openldap.org/doc/admin21/sasl.html

[7] Linuxtopia, "Open LDAP Administration – Table of Contents", http://www.linuxtopia.org/online_books/network_administration_guides/ldap_administration/

[8] International Telecommunication Union – Telecommunication Standardization Sector, "The Directory – Overview of concepts, models and services", X.500(1993)

[9] International Telecommunication Union – Telecommunication Standardization Sector, "The Directory – Models", X.501(1993)

[10] International Telecommunication Union – Telecommunication Standardization Sector, “The Directory: Abstract Service Definition”, X.511(1993)

[11] Philipp Föckeler, “LDAP – The Lightweight Directory Access Protocol”, www.selfadsi.org/ldap.html

[12] “Linked Lists”, http://www.learn-c.org/en/Linked_lists

[13] Display Filter Reference: “Lightweight Directory Access Protocol”, http://www.wireshark.org/docs/dfref/l/ldap.html

[14] Wireshark, “tshark – The Wireshark Network Analyzer”, http://www.wireshark.org/docs/man-pages/tshark.html

[15] “Tratamiento masivo de datos con AWK, alternativa parcial a ACL o SAS”, http://www.marblestation.com/?p=761#003

[16] Pilar Rodríguez, Escuela Politécnica Superior, Universidad Autónoma de Madrid, Teoría de Autómatas y Lenguajes Formales I, “Utilización de valgrind”, http://arantxa.ii.uam.es/~talf1/valgrind.pdf

[17] Wikipedia, The Free Encyclopedia, “Bit manipulation”, http://en.wikipedia.org/wiki/Bit_manipulation

[18] Lowell Heddings, “Using htop to Monitor System Processes on Linux”, http://www.howtogeek.com/howto/ubuntu/using-htop-to-monitor-system-processes-on-linux/

[19] About.com, Computing Linux, “Linux / Unix Command: time”, http://linux.about.com/library/cmd/blcmdl1_time.htm

[20] Maligno Alonso, “Un informático en el lado del mal: Ataques a LDAP”, http://www.elladodelmal.com/2008/06/ataques-ldap-i-de-iv-ataques-ldap-ii-de.html

[21] Emmanuel Lécharny, “LDAP ASN.1 Códec”, https://cwiki.apache.org/confluence/display/DIRxSRVx10/Ldap+ASN.1+Codec#LdapASN.1Codec-LDAPASN.1Grammar

A. Manual de instalación

1. Hacer “make” en la línea de comandos situado en la carpeta que contiene el código fuente del programa.

2. Ejecutar posteriormente el programa con:

   ./ldap_disector –f traza.pcap

B. Manual del programador

En el código fuente los ficheros principales del desarrollo del programa son ldap_disector.c y utils.c.

En ellos se hace referencia a todas las funciones que se han utilizado y es posible revisar el código de su implementación.

PRESUPUESTO

1) Ejecución Material

- Compra de ordenador personal (Software incluido)..... 2.000 €
- Alquiler de impresora láser durante 6 meses.....50 €
- Material de oficina .....150 €
- Total de ejecución material..... 2.200 €

2) Honorarios Proyecto

- 640 horas a 15 € / hora..... 9600 €

3) Material fungible

- Gastos de impresión..... 60 €
- Encuadernación..... 200 €

4) Subtotal del presupuesto

- Subtotal Presupuesto..... 12060 €

5) I.V.A. aplicable

- 21% Subtotal Presupuesto ..... 2532.6 €

6) Total presupuesto

- Total Presupuesto..... 14592,6 €

Madrid, Abril de 2014

El Ingeniero Jefe de Proyecto

Fdo.: Guillermo Muñoz Mozos

Ingeniero Superior de Telecomunicación

PLIEGO DE CONDICIONES

Este documento contiene las condiciones legales que guiarán la realización, en este proyecto, de un **Disector de Protocolos LDAP**. En lo que sigue, se supondrá que el proyecto ha sido encargado por una empresa cliente a una empresa consultora con la finalidad de realizar dicho sistema. Dicha empresa ha debido desarrollar una línea de investigación con objeto de elaborar el proyecto. Esta línea de investigación, junto con el posterior desarrollo de los programas está amparada por las condiciones particulares del siguiente pliego.

Supuesto que la utilización industrial de los métodos recogidos en el presente proyecto ha sido decidida por parte de la empresa cliente o de otras, la obra a realizar se regulará por las siguientes:

Condiciones generales

1. La modalidad de contratación será el concurso. La adjudicación se hará, por tanto, a la proposición más favorable sin atender exclusivamente al valor económico, dependiendo de las mayores garantías ofrecidas. La empresa que somete el proyecto a concurso se reserva el derecho a declararlo desierto.

2. El montaje y mecanización completa de los equipos que intervengan será realizado totalmente por la empresa licitadora.

3. En la oferta, se hará constar el precio total por el que se compromete a realizar la obra y el tanto por ciento de baja que supone este precio en relación con un importe límite si este se hubiera fijado.

4. La obra se realizará bajo la dirección técnica de un Ingeniero Superior de Telecomunicación, auxiliado por el número de Ingenieros Técnicos y Programadores que se estime preciso para el desarrollo de la misma.

5. Aparte del Ingeniero Director, el contratista tendrá derecho a contratar al resto del personal, pudiendo ceder esta prerrogativa a favor del Ingeniero Director, quien no estará obligado a aceptarla.

6. El contratista tiene derecho a sacar copias a su costa de los planos, pliego de condiciones y presupuestos. El Ingeniero autor del proyecto

autorizará con su firma las copias solicitadas por el contratista después de confrontarlas.

7. Se abonará al contratista la obra que realmente ejecute con sujeción al proyecto que sirvió de base para la contratación, a las modificaciones autorizadas por la superioridad o a las órdenes que con arreglo a sus facultades le hayan comunicado por escrito al Ingeniero Director de obras siempre que dicha obra se haya ajustado a los preceptos de los pliegos de condiciones, con arreglo a los cuales, se harán las modificaciones y la valoración de las diversas unidades sin que el importe total pueda exceder de los presupuestos aprobados. Por consiguiente, el número de unidades que se consignan en el proyecto o en el presupuesto, no podrá servirle de fundamento para entablar reclamaciones de ninguna clase, salvo en los casos de rescisión.

8. Tanto en las certificaciones de obras como en la liquidación final, se abonarán los trabajos realizados por el contratista a los precios de ejecución material que figuran en el presupuesto para cada unidad de la obra.

9. Si excepcionalmente se hubiera ejecutado algún trabajo que no se ajustase a las condiciones de la contrata pero que sin embargo es admisible a juicio del Ingeniero Director de obras, se dará conocimiento a la Dirección, proponiendo a la vez la rebaja de precios que el Ingeniero estime justa y si la Dirección resolviera aceptar la obra, quedará el contratista obligado a conformarse con la rebaja acordada.

10. Cuando se juzgue necesario emplear materiales o ejecutar obras que no figuren en el presupuesto de la contrata, se evaluará su importe a los precios asignados a otras obras o materiales análogos si los hubiere y cuando no, se discutirán entre el Ingeniero Director y el contratista, sometiéndolos a la aprobación de la Dirección. Los nuevos precios convenidos por uno u otro procedimiento, se sujetarán siempre al establecido en el punto anterior.

11. Cuando el contratista, con autorización del Ingeniero Director de obras, emplee materiales de calidad más elevada o de mayores dimensiones de lo estipulado en el proyecto, o sustituya una clase de fabricación por otra que tenga asignado mayor precio o ejecute con mayores dimensiones cualquier otra parte de las obras, o en general, introduzca en ellas cualquier modificación que sea beneficiosa a juicio del Ingeniero Director de obras, no tendrá derecho sin embargo, sino a lo que le correspondería si hubiera realizado la obra con estricta sujeción a lo proyectado y contratado.

12. Las cantidades calculadas para obras accesorias, aunque figuren por partida alzada en el presupuesto final (general), no serán abonadas sino a los precios de la contrata, según las condiciones de la misma y los proyectos particulares que para ellas se formen, o en su defecto, por lo que resulte de su medición final.

13. El contratista queda obligado a abonar al Ingeniero autor del proyecto y director de obras así como a los Ingenieros Técnicos, el importe de sus respectivos honorarios facultativos por formación del proyecto, dirección técnica y administración en su caso, con arreglo a las tarifas y honorarios vigentes.

14. Concluida la ejecución de la obra, será reconocida por el Ingeniero Director que a tal efecto designe la empresa.

15. La garantía definitiva será del 4% del presupuesto y la provisional del 2%.

16. La forma de pago será por certificaciones mensuales de la obra ejecutada, de acuerdo con los precios del presupuesto, deducida la baja si la hubiera.

17. La fecha de comienzo de las obras será a partir de los 15 días naturales del replanteo oficial de las mismas y la definitiva, al año de haber ejecutado la provisional, procediéndose si no existe reclamación alguna, a la reclamación de la fianza.

18. Si el contratista al efectuar el replanteo, observase algún error en el proyecto, deberá comunicarlo en el plazo de quince días al Ingeniero Director de obras, pues transcurrido ese plazo será responsable de la exactitud del proyecto.

19. El contratista está obligado a designar una persona responsable que se entenderá con el Ingeniero Director de obras, o con el delegado que éste designe, para todo relacionado con ella. Al ser el Ingeniero Director de obras el que interpreta el proyecto, el contratista deberá consultarle cualquier duda que surja en su realización.

20. Durante la realización de la obra, se girarán visitas de inspección por personal facultativo de la empresa cliente, para hacer las comprobaciones que se crean oportunas. Es obligación del contratista, la conservación de la obra ya ejecutada hasta la recepción de la misma, por lo

que el deterioro parcial o total de ella, aunque sea por agentes atmosféricos u otras causas, deberá ser reparado o reconstruido por su cuenta.

21. El contratista, deberá realizar la obra en el plazo mencionado a partir de la fecha del contrato, incurriendo en multa, por retraso de la ejecución siempre que éste no sea debido a causas de fuerza mayor. A la terminación de la obra, se hará una recepción provisional previo reconocimiento y examen por la dirección técnica, el depositario de efectos, el interventor y el jefe de servicio o un representante, estampando su conformidad el contratista.

22. Hecha la recepción provisional, se certificará al contratista el resto de la obra, reservándose la administración el importe de los gastos de conservación de la misma hasta su recepción definitiva y la fianza durante el tiempo señalado como plazo de garantía. La recepción definitiva se hará en las mismas condiciones que la provisional, extendiéndose el acta correspondiente. El Director Técnico propondrá a la Junta Económica la devolución de la fianza al contratista de acuerdo con las condiciones económicas legales establecidas.

23. Las tarifas para la determinación de honorarios, reguladas por orden de la Presidencia del Gobierno el 19 de Octubre de 1961, se aplicarán sobre el denominado en la actualidad “Presupuesto de Ejecución de Contrata” y anteriormente llamado ”Presupuesto de Ejecución Material” que hoy designa otro concepto.

Condiciones particulares

La empresa consultora, que ha desarrollado el presente proyecto, lo entregará a la empresa cliente bajo las condiciones generales ya formuladas, debiendo añadirse las siguientes condiciones particulares:

1. La propiedad intelectual de los procesos descritos y analizados en el presente trabajo, pertenece por entero a la empresa consultora representada por el Ingeniero Director del Proyecto.

2. La empresa consultora se reserva el derecho a la utilización total o parcial de los resultados de la investigación realizada para desarrollar el siguiente proyecto, bien para su publicación o bien para su uso en trabajos o proyectos posteriores, para la misma empresa cliente o para otra.

3. Cualquier tipo de reproducción aparte de las reseñadas en las condiciones generales, bien sea para uso particular de la empresa cliente, o para cualquier otra aplicación, contará con autorización expresa y por escrito del Ingeniero Director del Proyecto, que actuará en representación de la empresa consultora.

4. En la autorización se ha de hacer constar la aplicación a que se destinan sus reproducciones así como su cantidad.

5. En todas las reproducciones se indicará su procedencia, explicitando el nombre del proyecto, nombre del Ingeniero Director y de la empresa consultora.

6. Si el proyecto pasa la etapa de desarrollo, cualquier modificación que se realice sobre él, deberá ser notificada al Ingeniero Director del Proyecto y a criterio de éste, la empresa consultora decidirá aceptar o no la modificación propuesta.

7. Si la modificación se acepta, la empresa consultora se hará responsable al mismo nivel que el proyecto inicial del que resulta el añadirla.

8. Si la modificación no es aceptada, por el contrario, la empresa consultora declinará toda responsabilidad que se derive de la aplicación o influencia de la misma.

9. Si la empresa cliente decide desarrollar industrialmente uno o varios productos en los que resulte parcial o totalmente aplicable el estudio de este proyecto, deberá comunicarlo a la empresa consultora.

10. La empresa consultora no se responsabiliza de los efectos laterales que se puedan producir en el momento en que se utilice la herramienta objeto del presente proyecto para la realización de otras aplicaciones.

11. La empresa consultora tendrá prioridad respecto a otras en la elaboración de los proyectos auxiliares que fuese necesario desarrollar para dicha aplicación industrial, siempre que no haga explícita renuncia a este hecho. En este caso, deberá autorizar expresamente los proyectos presentados por otros.

12. El Ingeniero Director del presente proyecto, será el responsable de la dirección de la aplicación industrial siempre que la empresa consultora lo estime oportuno. En caso contrario, la persona designada deberá contar con la autorización del mismo, quien delegará en él las responsabilidades que ostente.